

A Methodology for the Efficient Integration of Transient Constraints in the Design of Aircraft Dynamic Systems

A Thesis
Presented to
The Academic Faculty

by

Léon L. Phan

In Partial Fulfillment
of the Requirements for the Degree
Doctorate of Philosophy

School of Aerospace Engineering
Georgia Institute of Technology
August 2010

Copyright © 2010 by Léon L. Phan

A Methodology for the Efficient Integration of Transient Constraints in the Design of Aircraft Dynamic Systems

Approved by:

Prof. Dimitri N. Mavris
Committee Chair
School of Aerospace Engineering,
Georgia Institute of Technology

Prof. Santiago Grijalva
School of Electrical and Computer En-
gineering,
Georgia Institute of Technology

Dr. Elena Garcia
School of Aerospace Engineering,
Georgia Institute of Technology

Prof. Daniel Schrage
School of Aerospace Engineering,
Georgia Institute of Technology

Dr. Jean-Jacques Charrier
R&T Division,
Hispano-Suiza, France

Date Approved: May 20, 2010

To my parents, for everything, and to Carolina and K  vin. For too long, I have been a long-distance son and brother.

ACKNOWLEDGEMENTS

At the end of this long journey towards my Ph.D., spent within the Aerospace Systems Design Laboratory, I realize that there are many people who have made it possible for me to climb the mountain. I feel privileged and grateful, and I would like to thank them all.

Firstly, I would like to express my profound gratitude to my research advisor, Dr. Dimitri Mavris, for his support and for all the opportunities that ASDL has offered me. Doc, I have learnt so much and feel that I have grown, thanks to your tireless dedication towards the personal and educational development of your students. Merci Doc.

I am also deeply grateful to Dr. Elena Garcia, who has closely followed my research throughout my years at ASDL, with dedication and genuine enthusiasm. Her eternal good mood made it a pleasure to work with her.

My special thanks also go to Dr. Jean-Jacques Charrier from Hispano-Suiza, who has continuously supported my research and who has, as a result, spent many hours on transatlantic flights reading my thesis material.

I would also like to acknowledge the other members of my thesis committee, Dr. Santiago Grijalva and Dr. Daniel Schrage, for their valuable insights and comments on the topics of power systems and systems engineering.

My thesis was sponsored by Hispano-Suiza (Safran Group). I am grateful to the company and its employees, in particular to Jean-Yves Routex, Nathalie Chantelat, Philippe Poret, Marianne Félix and all the COPPER Bird team, for their time, their effort and commitment, their valuable technical insights, as well as the kindness

they have showed me, especially during my three month stay in Colombes with the COPPER Bird team.

During my stay at ASDL, I was surrounded by dynamic, enthusiastic and talented people who contribute to making this place unique, and I have made many friends there. I would like to thank all the staff, with special thanks to Cara Zell and Loretta Carroll. I would like to thank my colleagues, in particular Stéphane Dufresne, Cyril de Tenorio, Olivia Pinon, Kyle Collins, Frédéric Villeneuve, Simon Briceño, Ismael Fernandez, Eric Upton, Peter Hollingsworth, Peter Schmollgruber, Jesse Eyer, Mandy Goltsch, Marcus Smith, Michael Armstrong, Michael Balchanos, Ousmane Diallo, Bassem Nairouz, Cédric Justin, Alexia Payan, and Dongwook Lim. Thank you for your support and for having made my stay at ASDL such an enjoyable one.

It is with great emotion that I thank all my friends from outside ASDL, who have brought me balance and strength all through these years. First, I want to thank Cyril, Jean-Francois, and the Atlanta Symphony Orchestra for the wonderful musical moments that we have shared. I would like to thank the MIT French community: Nathalie, Jeru, and Daniel in particular. I want to thank the Spanish crew for their warm embrace and contagious good mood (*paquito en la playa*): muchas gracias a todos, especialmente a Esperanza, Mari-Carmen, David, Juan-Jo, Quim, Benji y Carmen. I would also like to thank the close guard, the French mafia, and its affiliates: Igor, Gilles, Rémi Martoni, Stephani, Reah, Nathalie, Chlutz, Alpha, Stéphane, Jen, Florent, Aurelija, Pedro, Yannick, Callie, Matthieu, and Hélène. Thank you for all the memorable moments, and for your support during tough times.

I would like to give a warm thank you to my two dearest friends, Franklin and Diane, who have been almost like a family to me, and who have made it possible for me to cope with difficult moments. Merci bande de collocs! Sans vous, je n'aurais pas tenu la distance.

Finally, I want to thank my friends and family in France who have supported me and sent me their good vibrations across the Atlantic. First, I am infinitely grateful to my parents, who have always spent all their energy to ensure that I could follow whatever path I would choose. Tres chers parents, cette these est pour vous. I would like to thank Carolina and Kévin for their encouragements, and I hope they forgive the prolonged absence of their big brother. My final thought goes to the treasure that life has given me, my favorite Andalucian, Teresa. Mi Amor, tu m'as tant donné. Gracias. Sans toi, je n'aurais pas tenu le coup lors du sprint final. Maintenant, du sommet, je vois les horizons, ils sont vastes, et nous appartiennent.

Contents

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	xi
LIST OF FIGURES	xii
NOMENCLATURE	xviii
SUMMARY	xxii
I INTRODUCTION	1
1.1 Dynamic Systems and Transient Regimes	2
1.2 Relevance of Transient Regime Considerations in Aircraft Design	5
1.3 Transient Constraints in Design: Dynamic Constraints	9
1.4 Aircraft Development Process	10
1.4.1 Specification and Verification	10
1.4.2 Simulation: an Analysis Tool for Verification	14
1.4.3 Simulation as a Tool for Specification	16
1.4.4 Uncertainty in the Development Process	17
1.5 Dynamic Systems Development Process	18
1.5.1 Time-Domain Simulation: First Introduction	19
1.5.2 Limitations of Time-Domain Simulation and Consequences	20
1.5.3 Controlled Dynamic Systems	21
1.5.4 Dynamic Constraints: Challenges for the Design of Dynamic Systems	25
1.6 Summary and Research Objectives	27
1.7 Organization of the Dissertation	28
II ADVANCED DESIGN METHODS FOR DYNAMIC SYSTEMS AND FIRST HYPOTHESES	30
2.1 Design Space Exploration of Dynamic Systems	30

2.1.1	Design Space and Operation Space	30
2.1.2	Optimization-based Methods	33
2.1.3	Data Farming for Design Space Exploration	43
2.1.4	An Innovative Data Farming Approach for the Design of Dynamic Systems: Time-Domain Filtered Monte-Carlo	58
2.2	Advanced Modeling and Simulation of Dynamic Systems	71
2.2.1	Modeling of Dynamic Systems	71
2.2.2	Methods for Time-Domain Simulation	78
2.2.3	Surrogate Modeling for Efficient Simulation	83
2.3	Summary: First Elements of the Methodology	94
III	SYSTEM IDENTIFICATION: CLASSIFICATION AND BENCHMARKING	97
3.1	Characterization of the Problem	97
3.1.1	Number of Parameters	97
3.1.2	Response Linearity	98
3.1.3	Black Box vs. Grey Box Modeling	99
3.1.4	Static Nature of Input Parameters	99
3.1.5	Multiscale Nature of the Responses	101
3.1.6	Simulation Time and Transient Perturbation Time	103
3.1.7	Summary and Desired Characteristics	104
3.2	Benchmarking	104
3.2.1	Feedforward Neural Networks	105
3.2.2	Recurrent Neural Networks	108
3.2.3	Meijer's Dynamic Neural Networks	111
3.2.4	Wavelet Neural Networks	115
3.2.5	Benchmarking Summary and Alternative System Identification Formulation	124
3.3	Contingency Plan: Formulation of an Alternate Hypothesis	125
3.4	Summary	127

IV	ENVELOPE DETECTION METHODS FOR ENVELOPE IDENTIFICATION	129
4.1	Simple Methods for Envelope Detection	131
4.1.1	Sliding Window	131
4.1.2	Hilbert Transform Based Envelope Detection	133
4.2	Envelope Detection Method Based on Multiresolution Analysis with Wavelets	134
4.2.1	Multiresolution Analysis	134
4.2.2	Envelope Detection Method Based on MRA Decompositions	138
4.3	Summary: MRA-Based Envelope Detection Method for Envelope Identification	139
V	HYPOTHESES AND METHODOLOGY FORMULATION	142
5.1	Recapitulation	142
5.2	Methodology	144
5.2.1	Step 1: Define the Problem	144
5.2.2	Step 2: Create the Modeling and Simulation Environment	144
5.2.3	Step 3: Test Planning - Design of Experiments	146
5.2.4	Step 4: Test Campaign - Collect the Training Data	148
5.2.5	Step 5: Create Dynamic Surrogate Models - Train the Neural Networks	152
5.2.6	Step 6: Perform a Monte-Carlo Simulation Using the Dynamic Surrogate Models	161
5.2.7	Step 7: Populate the Interactive Visualization Environment	162
5.2.8	Step 8: Explore and Filter the Design/Operation Space	164
5.2.9	Summary	164
VI	METHODOLOGY IMPLEMENTATION AND PROOF OF CONCEPT	167
6.1	Experiment 1: System Identification of a Simple Mathematical System	167
6.1.1	Objectives and Plan	167
6.1.2	Implementation and Results	169

6.1.3	Summary	181
6.2	Experiment 2: Visual Transient Response Exploration	181
6.2.1	Objectives and Plan	181
6.2.2	Implementation and Results	183
6.3	Experiment 3: Proof of Concept - Implementation of the Methodology for the Design of an Electrical Network	195
6.3.1	Objectives and Plan	196
6.3.2	Implementation and Results	199
6.3.3	Summary and Methodology Validation	225
VII	CONCLUDING REMARKS	228
7.1	Recapitulation	228
7.2	Contributions	231
7.3	Limitations and Recommendations for Future Work	233
Appendix A	— WAVELET THEORY: BACKGROUND	236
Appendix B	— BACKPROPAGATION ALGORITHM	238
Appendix C	— MATLAB SOURCE CODE FOR WAVENET TRAIN- ING	240
Appendix D	— JSL SCRIPT FOR THE GENERATION OF VISTRE249	249
	REFERENCES	277
	VITA	289

List of Tables

Table 1	Examples of Dynamic and Static Parameters	3
Table 2	Thrust-to-Weight Ratio Estimation [129]	33
Table 3	Design Options for Visualization of Time-Domain Data [5]	63
Table 4	Classification of System Models [89]	74
Table 5	Feedforward Neural Network Assessment	108
Table 6	Recurrent Neural Network Assessment	111
Table 7	Meijer’s Dynamic Neural Network Assessment	115
Table 8	Wavenet Assessment	124
Table 9	Benchmarking Summary	124
Table 10	2D Wavenet - Table of Coefficients	171
Table 11	2D Wavenet - MFE and MRE Statistics	175
Table 12	DoE Table for the Tri-Dimensional Mathematical System	178
Table 13	JMP Monte-Carlo Signal Table (MCST) for Response Y^k	185
Table 14	JMP Monte-Carlo Main Table (MCMT)	185
Table 15	JMP Y^k _Aggregate Table	186
Table 16	JMP Y^k _Aggregate Table after Adding Constraint C	193
Table 17	Design Variables for the 350VDC Electrical Network	200

List of Figures

Figure 1	Examples of Transient Events Experienced by Aircraft Systems . . .	4
Figure 2	Power Quality Impact of Load Switching: Voltage Sag	6
Figure 3	POA Architecture Concept [52]	7
Figure 4	Transient Input Phase Current to an Aircraft Flight Control Surface Actuation System during a Typical Actuation Maneuver [55]	8
Figure 5	Transient Voltage Constraint for 350VDC Networks [52]	9
Figure 6	Aircraft Development Process: Waterfall View [36]	11
Figure 7	Design of Level N Subsystem	12
Figure 8	Aircraft Development Process: V-Diagram [13]	13
Figure 9	V-Diagram and Virtual Verification [13]	14
Figure 10	Simulation-based Design: Elementary Approach	16
Figure 11	Time-Domain Simulation Process	19
Figure 12	Open-loop Control System	22
Figure 13	Closed-loop Control System	23
Figure 14	Controller Design Process	24
Figure 15	Design of Level N Subsystem with Dynamic Constraints	26
Figure 16	Design Space Exploration: Wing Planforms [20]	31
Figure 17	Design and Operation Spaces	32
Figure 18	Generic Path-Searching Optimization Algorithm	36
Figure 19	Transient Static Parameters for a Typical Response to a Step Input [91]	37
Figure 20	Weighting Factor $K(t)$ for Transient Regime [51]	39
Figure 21	Simulation-Based Optimization for a Transient Controller [51] . . .	40
Figure 22	Local and Global Minima	41
Figure 23	Visual Analytics for Typhoon Monitoring [156, 137]	46
Figure 24	Heat Map for Genomics [131]	47
Figure 25	Correlation Matrix for Genome Matching [131]	48

Figure 26	Three-dimensional Visualization of a CFD Model [163]	48
Figure 27	Linked Interactive Visualizations [131]	49
Figure 28	Large Displays for Collaborative Visualization Environment [42] . .	50
Figure 29	Data Farming Process	51
Figure 30	Monte-Carlo Simulation Process	53
Figure 31	Scatterplot Matrix for the Filtered Monte-Carlo Technique	55
Figure 32	Scatterplot Matrix with Constraints for the Filtered Monte-Carlo Technique	57
Figure 33	Scatterplot Matrix after Filtering for the Filtered Monte-Carlo Tech- nique	57
Figure 34	Time-Domain Filtered-Monte-Carlo: First Description of the Visual Environment	60
Figure 35	Three Dimensional Visual Representation of Power Consumption [155]	61
Figure 36	Clustering and Calendar Based Representation of Signals [155] . . .	62
Figure 37	TimeSearcher Visual Interface	62
Figure 38	Spiral Visualization of Power Consumption [154, 98]	63
Figure 39	Overlay Plot for One Time-Domain Cell of VisTRE	65
Figure 40	VisTRE: Visualization of One Design Scenario	66
Figure 41	VisTRE: Visualization of the Full Design Space	67
Figure 42	VisTRE: Visualization of Undesirable Design Scenarios	68
Figure 43	VisTRE: Visualization of the Design Space after Filtering	68
Figure 44	Using VisTRE for Design Space Exploration of Dynamic Systems: Algorithm	69
Figure 45	Time-Domain Simulation: Settling Time Before Perturbation . . .	70
Figure 46	Block Diagram of a State-Space Simulation [89]	76
Figure 47	Approximation Error of the Euler Integration Method [89]	80
Figure 48	Steps in a Simulation Study [92]	82
Figure 49	Static Surrogate Model	84
Figure 50	Generic Process for the Generation of a Surrogate Model	86
Figure 51	Schematics of a Neuron with Transfer Function φ	88

Figure 52	Sigmoid Function	89
Figure 53	Example of Artificial Neural Network Architecture with One Hidden Layer	90
Figure 54	Dynamic Surrogate Model	93
Figure 55	Time-Domain Filtered-Monte-Carlo: First Overview	96
Figure 56	Transient Perturbations for the Analysis of an Electrical Test Rig [127]	100
Figure 57	Characterization of a Dynamic Ramp with Static Parameters	100
Figure 58	Dynamic surrogate model with static variables	101
Figure 59	Decomposition of a Multiscale Signal	102
Figure 60	Steady-state Voltage of a 350 VDC Network [52]	103
Figure 61	Time-Domain Feedforward Neural Network	106
Figure 62	Sine Wave	107
Figure 63	Feedforward Neural Network for Approximating a Simple Time Series	107
Figure 64	Hopfield Network with Four Neurons [130]	109
Figure 65	Example of Recurrent Neural Network Architecture	110
Figure 66	Schematics of Meijer’s Dynamic Neural Networks [109]	113
Figure 67	Mexican Hat Wavelet and Morlet Wavelet	117
Figure 68	Dilation and Translation of the Mother Wavelet	118
Figure 69	3D Gabor Wavelet [31]	120
Figure 70	Schematic of a Wavelon with Mother Wavelet ψ	121
Figure 71	Wavenet Architecture	122
Figure 72	Signal Envelope	126
Figure 73	Equivalent System Model for Envelope Detection	126
Figure 74	Schematics of an Envelope Detection Scheme	130
Figure 75	Amplitude Modulated Signal with a Sine Wave Carrier	131
Figure 76	Sliding Window Method	131
Figure 77	Sliding Window Method with Smaller Window Size	132
Figure 78	Frequency Spectrum of a Bandpass Signal	134
Figure 79	Dyadic Grid of Scaling and Translation Factors in a MRA	135

Figure 80	Daubechies Wavelet and Scaling Functions at the Order 4 [105]	136
Figure 81	Multiresolution Analysis Decomposition Tree [105]	137
Figure 82	Signal Denoising	138
Figure 83	Envelope Detection Method Based on MRA	139
Figure 84	Envelope Identification Process	141
Figure 85	Envelope Reconstruction with Dynamic Surrogate Models	141
Figure 86	Random Latin Hyper Cube Design for Two Dimensions [12]	147
Figure 87	Optimal Latin Hyper Cube Design for Two Dimensions [12]	148
Figure 88	Test Campaign Output for the Dynamic Response Y^k	149
Figure 89	Training Data Collection for the Dynamic Response Y^k	150
Figure 90	Subsampling of the Trend Signal	151
Figure 91	Training Matrix for the Trend Signal of the Dynamic Response Y^k	152
Figure 92	Wavenet Architecture	154
Figure 93	Wavenet Training Process	155
Figure 94	Dynamic Feedforward Neural Network	156
Figure 95	Neural Network Training Process	157
Figure 96	Illustration of Overfitting	159
Figure 97	Actual vs. Predicted	160
Figure 98	Residual vs. Predicted	161
Figure 99	Desired Distribution Shape for MFE and MRE	161
Figure 100	Envelope Reconstruction after the Monte-Carlo Simulation	163
Figure 101	VisTRE Populated with the Outputs of the Monte-Carlo Simulation	163
Figure 102	Methodology Overview	166
Figure 103	Representation of the Simple Dynamic System	168
Figure 104	2D Wavenet - Actual vs. Predicted	172
Figure 105	2D Wavenet - Residual vs. Predicted	173
Figure 106	2D Wavenet - Residual vs. Time	174
Figure 107	2D Wavenet - MFE for the Relative Error	175
Figure 109	2D Wavenet - MRE for the Residual Error	176

Figure 108 2D Wavenet - MFE for the Residual Error	176
Figure 110 LHC DoE Table for the Training of the 3D System	178
Figure 111 3D Wavenet - Actual vs. Predicted	179
Figure 112 3D Wavenet - Residual vs. Time	180
Figure 113 3D Wavenet - Evolution of the Total Regression Error During Training	180
Figure 114 JMP Data Table	184
Figure 115 Interaction between the Y^k _Aggregate Table and its Overlay Plot	187
Figure 116 Interaction between the MCMT and the Scatterplot	188
Figure 117 VisTRE: Scatterplot for Static Parameters	189
Figure 118 VisTRE: Overlay for Dynamic Transient Response (Voltage)	189
Figure 119 VisTRE: Highlighting Signals and Design Points	191
Figure 120 VisTRE: Adding a Dynamic Constraint for a Dynamic Response .	192
Figure 121 VisTRE: Finding the Points Violating the Dynamic Constraint . .	194
Figure 122 VisTRE: Filtering the Signals the Meet the Dynamic Constraint . .	194
Figure 123 Simulink Diagram of the 350VDC Network	196
Figure 124 Simulink Diagram of the Controlled 350VDC Generator	197
Figure 125 Block Diagram of a PI Controller	198
Figure 126 Simulink Diagram of the Actuator Motor	198
Figure 127 Power Quality Transient Constraint for the 350VDC Network Voltage	199
Figure 128 Generator Rotor Speed Behavior	202
Figure 129 Network Voltage Behavior	202
Figure 130 Network Voltage: Focus on the Transient Region	203
Figure 131 MRA at Level 10 on the Voltage Response	204
Figure 132 Test Campaign: Simulation Run Times and Post-processing Times	205
Figure 133 BRAINN User Interface [74]	206
Figure 134 Goodness of Fit for $\ln(loss)$	207
Figure 135 Goodness of Fit for V_{trend}	208
Figure 136 Goodness of Fit for $V_{rippleInf}$	209
Figure 137 Goodness of Fit for $V_{rippleSup}$	210

Figure 138 VisTRE: Scatterplot Matrix for Design Variables and Loss	214
Figure 139 VisTRE: Overlay Plot for Vinf	215
Figure 140 VisTRE: Maximum Loss Points	216
Figure 141 VisTRE: Vinf Behavior for the 4 Maximum Loss Points	217
Figure 142 VisTRE: Overlay of Vinf with Minimum Constraint	217
Figure 143 VisTRE: Finding the Points Violating the Constraint in the Vinf Overlay	218
Figure 144 VisTRE: Selecting the Full Signals Violating the Constraints in the Vinf Overlay	219
Figure 145 VisTRE: Filtered Scatterplot	220
Figure 146 VisTRE: Filtered Scatterplot with Region of Comfort	221
Figure 147 VisTRE: Filtered Scatterplot with Highlighted Region of Comfort .	222
Figure 148 VisTRE: Filtered Overlay with Hidden Undesirable Region for Vinf	223
Figure 149 VisTRE: Filtered Scatterplot with Hidden Undesirable Region for Vinf	224
Figure 150 VisTRE: Filtered Scatterplot with Minimum Loss Points	225
Figure 151 Loss Constraint in the Scatterplot for the Original Filtered-Monte- Carlo Method	226
Figure 152 Loss Constraint in the VisTRE Scatterplot for the Time-Domain Filtered-Monte-Carlo	227
Figure 153 Methodology Overview	232
Figure 154 Error Backpropagation for Neuron N_1	239

NOMENCLATURE

ACM	Air Cycle Machine
AM	Amplitude Modulation
ANN	Artificial Neural Network
ASVR	Aircraft System Validation Rig
BRAINN	Basic Regression Analysis for Integrated Neural Networks
CFD	Computational Fluid Dynamics
CWT	Continuous Wavelet Transform
DHS	Department of Homeland Security
DoE	Design of Experiments
DWT	Discrete Wavelet Transform
EHA	Electrohydraulic Actuator
EMI	Electromagnetic Interference
FFNN	Feedforward Neural Network
FWT	Fast Wavelet Transform
GA	Genetic Algorithm
GCU	Generator Control Unit
IDWT	Inverse Discrete Wavelet Transform
IGBT	Insulated-Gate Bipolar Transistor

INCOSE	International Council on Systems Engineering
ISS	International Space Station
IWT	Inverse Wavelet Transform
KDD	Knowledge Discovery in Databases
LCC	Life Cycle Cost
LTI	Linear Time-Invariant
M&S	Modeling and Simulation
MCMT	Monte-Carlo Main Table
MCS	Monte-Carlo Simulation
MCST	Monte-Carlo Signal Table
MDNN	Meijer's Dynamic Neural Network
MEA	More-Electric Aircraft
MFE	Model Fit Error
MIMO	Multiple Inputs Multiple Outputs
MISO	Multiple Inputs Single Output
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MRA	Multi-Resolution Analysis
MRE	Model Representation Error
MSE	Mean Squared Error
ODE	Ordinary Differential Equation

OEC	Overall Evaluation Criterion
OLHC	Optimal Latin Hyper Cube
PDE	Partial Differential Equation
PI	Proportional-Integral
PL	Programmable Load
PMG	Permanent Magnet Generator
POA	Power Optimised Aircraft
RF	Radio Frequency
RF	Recirculation Fan
RLHC	Random Latin Hyper Cube
RNN	Recurrent Neural Network
RSE	Response Surface Equation
SLS	Sea Level Static
SSE	Sum of Squared Errors
SST	Total Sum of Squares
TD-FMC	Time-Domain Filtered-Monte-Carlo
TDS	Time Domain Simulation
TOGW	Take-Off Gross Weight
VAC	Volt Alternative Current
VDC	Volt Direct Current

VisTRE	Visual Transient Response Explorer
WNN	Wavelet Neural Network (or Wavelet Network, or Wavenet)
WT	Wavelet Transform

SUMMARY

The motivation behind this thesis mainly stems from previous work performed at Hispano-Suiza (Safran Group) in the context of the European research project “Power Optimised Aircraft”. Extensive testing on the COPPER Bird[®], a test rig designed to characterize aircraft electrical networks, demonstrated the relevance of transient regimes in the design and development of dynamic systems.

Transient regimes experienced by dynamic systems may have severe impacts on the operation of the aircraft. For example, the switching on of a high electrical load might cause a network voltage drop inducing a loss of power available to critical aircraft systems. These transient behaviors are thus often regulated by dynamic constraints, requiring the dynamic signals to remain within bounds whose values vary with time. The verification of these peculiar types of constraints, which generally requires high-fidelity time-domain simulation, intervenes late in the system development process, thus potentially causing costly design iterations.

The research objective of this thesis is to develop a methodology that integrates the verification of dynamic constraints in the early specification of dynamic systems. In order to circumvent the inefficiencies of time-domain simulation, multivariate dynamic surrogate models of the original time-domain simulation models are generated, building on a nonlinear system identification technique using wavelet neural networks (or wavenets), which allow the multiscale nature of transient signals to be captured.

However, training multivariate wavenets can become computationally prohibitive as the number of design variables increases. Therefore, an alternate approach is formulated, in which dynamic surrogate models using sigmoid-based neural networks are used to emulate the transient behavior of the envelopes of the time-domain response.

Thus, in order to train the neural network, the envelopes are extracted by first separating the scales of the dynamic response, using a multiresolution analysis (MRA) based on the discrete wavelet transform. The MRA separates the dynamic response into a trend and a noise signal (ripple). The envelope of the noise is then computed with a windowing method, and recombined with the trend in order to reconstruct the global envelope of the dynamic response.

The run-time efficiency of the resulting dynamic surrogate models enable the implementation of a data farming approach, in which a Monte-Carlo simulation generates time-domain behaviors of transient responses for a vast set of design and operation scenarios spanning the design and operation space. An interactive visualization environment, enabling what-if analyses, will be developed; the user can thereby instantaneously comprehend the transient response of the system (or its envelope) and its sensitivities to design and operation variables, as well as filter the design space to have it exhibit only the design scenarios verifying the dynamic constraints.

The proposed methodology, along with its foundational hypotheses, are tested on the design and optimization of a 350VDC network, where a generator and its control system are concurrently designed in order to minimize the electrical losses, while ensuring that the transient undervoltage induced by peak demands in the consumption of a motor does not violate transient power quality constraints.

Chapter I

INTRODUCTION

This thesis deals with the design of aircraft dynamic systems that experience transient regimes during flight. This chapter gives some background on the topic and elements of motivation for this work, which will result in the formulation of an overall research objective.

The motivation behind this thesis mainly stems from previous work performed at Hispano-Suiza (Safran Group) in the context of a project called the Power Optimised Aircraft (POA), a research effort on the topic of More-Electric Aircraft (MEA), conducted by the major stakeholders of the European aeronautical industry, and funded by the European Union [3]. In the final stages of the POA program, innovative electrically powered system prototypes were developed, integrated, and tested in the COPPER Bird[®], the electrical test bench at Hispano-Suiza. The result of the integration of the POA system prototypes in the COPPER Bird[®] formed the Aircraft System Validation Rig (ASVR).

In this context, Phan et al. generated computer models of the behavior of the ASVR after performing multiple runs on the actual Hispano-Suiza test bench [127]. This activity enabled the authors to gain valuable insights on the relevance of transient regimes for design considerations.

As will be elaborated in this chapter, the study of transient regimes on the ASVR highlighted the fact that transient regimes play a critical role in the development of aircraft dynamic systems. In addition to lessons learned during the POA program and the corresponding work on the COPPER Bird[®], the next sections present the results of a preliminary literature review on the role of transient regimes in the design

of aircraft dynamic systems. The identified challenges of current design processes will serve as motivation for this thesis and its research objective.

1.1 Dynamic Systems and Transient Regimes

This section sets the stage for this thesis by defining some of the essential concepts that will be employed hereafter.

Aircraft are becoming more and more complex, as they consist of ever more highly sophisticated components that form higher-level *systems*. In literature, there exist numerous definitions of the word “system”. In order to avoid any ambiguity, the definition adopted herein is that of the International Council on System Engineering (INCOSE)[2] :

A system is “an integrated set of elements to accomplish a defined objective. These include hardware, software, firmware, people, information, techniques, facilities, services, and other support elements.”

This very broad definition encompasses concepts as diverse as groups of people or software, as long as they serve the fulfillment of a common objective. In the context of aircraft design, the objective is usually called “*function*”. The aircraft can be viewed as a system constituted by other systems dedicated to the fulfillment of the major functions, such as the engine, the wings, and the electrical power system. These systems are themselves comprised of lower level systems (or subsystems), and so on until the end of the system breakdown tree is reached. In this thesis, a system will refer to any system or subsystem within the system tree.

During its operation, a system can be described by its state variables (or simply “its *states*”). For example, an aircraft aileron’s operations may be described by the deflection angles taken by the control surface. Most aircraft systems are *dynamic*, i.e. their states change with time [8]. On the contrary, a system whose states remain constant is called *static*.

This opposition between “dynamic” and “static” is also applicable to parameters in general. A *dynamic parameter* (or *dynamic signal*, or *signal*) will see its value evolve with time while a *static parameter* will not. For example, the voltage of a perfect sine voltage generator is a dynamic parameter, while the amplitude and frequency of the sine voltage are static parameters. Another illustration of the concepts of static and dynamic parameters are given by the weight of the aircraft. As the aircraft flies its mission and burns fuel, the overall aircraft weight decreases. Therefore, the aircraft weight is a dynamic parameter. However, the maximum takeoff gross weight or the zero fuel empty weights are static parameters. Table 1 lists examples of dynamic parameters and related static ones. Formally, a dynamic parameter can thus be defined as a function of time $y : t \mapsto y(t)$ while a static parameter is a real (or complex) value $y \in \mathbb{R}$ (or \mathbb{C}).

Table 1: Examples of Dynamic and Static Parameters

Dynamic Parameters	Static Parameters
aircraft weight	maximum takeoff gross weight
engine thrust	maximum engine sea level static thrust
periodical voltage	frequency of the periodical voltage
power delivered by the generator	maximum power of the generator

During their operation, dynamic systems generally go through a variety of regimes, ranging from equilibrium or stationary points, which are called “*steady-state regimes*”, to short and abrupt variations of state, which are termed “*transient regimes*”. It is important to note that while transient regimes are by nature described by states that are dynamic, a dynamic state may not necessarily indicate that the system lies outside a steady-state regime. Indeed, a system whose state is dynamic but periodic may be considered as in steady-state [136]. This is due to the periodicity of the signal, whereby any dynamic parameter or state describing a steady-state regime of this signal can be defined with a finite collection of static parameters, such as variation amplitude, frequency, and Fourier coefficients. For instance, an electric generator

whose voltage output follows a sine wave is in steady-state. Its voltage is a dynamic signal that can be fully described by two static parameters: the sine amplitude and frequency.

Throughout an aircraft mission, transient regimes occur often and may affect many systems. Examples of transient events that may occur during operation are shown in Figure 1. When the engine starts, the speed of the engine shaft goes from 0 RPM to thousands of RPM's in less than a second, thus inducing a transient perturbation on the electrical network that it drives. The electrical system also experiences transient events due to switching of loads (such as onboard cabin equipment being turned on or off, or the sudden shutdown of an engine) or exterior events (such as lightning). When an aileron or a spoiler is deflected, the aerodynamic load varies abruptly, thereby experiencing a transient state before reaching a new equilibrium. Another source of transients is the actuation of the thrust reverse system at landing, which creates transient thermal and mechanical stress on the nacelle as well as electrical transient perturbations of the electrical system. In the next section, it will be established that transient regimes may have serious consequences on the operation of the aircraft and are therefore of prime importance when designing dynamic systems.

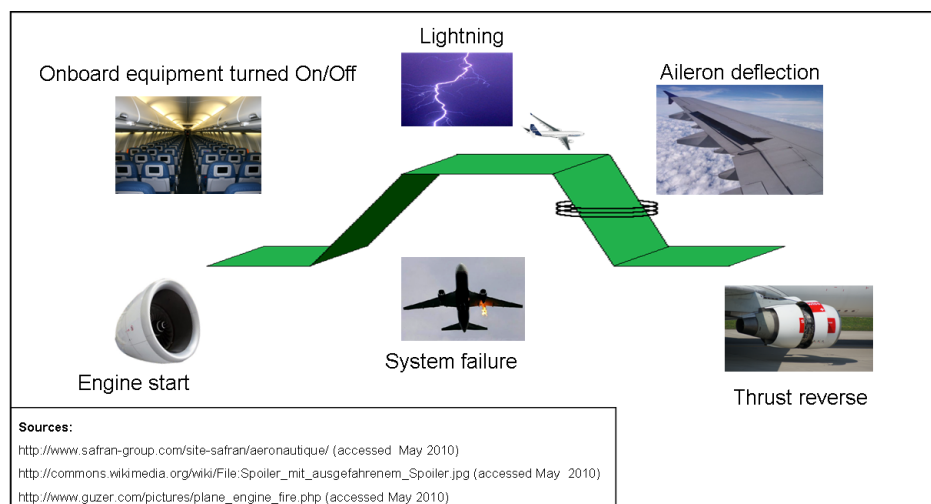


Figure 1: Examples of Transient Events Experienced by Aircraft Systems

1.2 Relevance of Transient Regime Considerations in Aircraft Design

As aircraft system architectures grow in complexity, it has become essential to improve the design and development process. Because the bulk of the Life Cycle Cost (LCC) of the aircraft depends on the decisions made in the early stages of the design, one of the most potentially rewarding research efforts deals with making the conceptual and preliminary design phases more efficient: one area of such efforts focuses on the integration of subsystem design considerations into these early design stages [4].

The design of a subsystem (or simply “system”) derives from a set of operations that the system will be expected or required to perform. Throughout the aircraft mission, most dynamic systems experience transient regimes. One obvious example of transient regime is the oscillatory response of the aircraft body incurred by a step command given by the pilot to the aileron. Transient regimes experienced by a system may have significant impacts on the operation of other systems. For example, a sudden and heavy increase in electrical power consumption from a system equates to an increase in the power required from the electrical generator. If the electrical generator is driven by the engine shaft, this variation of power demand may induce a short oscillatory variation of torque on the engine shaft, which in turn may affect the level of available thrust. Not only is this undesirable, but it can even reduce the surge margin to a point where the engine might fail during a transient maneuver[27].

Transient regimes may be induced by external and uncontrollable events. For instance, lightning may cause serious transient disturbances on the electrical network. These disturbances, which are a particular case of Electromagnetic Interference (EMI), can affect flight critical systems, and as documented in Clarke et al., they could lead, in the worst scenarios, to engine shutdowns and fatal crashes[25]. EMI relates to the growing field of power quality assurance, which defines a set of standards and rules that the behavior of the electrical network should respect in order for

the electrical equipment to remain operational. Power quality requirements govern steady-state regimes as well as transient ones. Other relevant examples of transient power quality issues besides EMI pertain to the level of network voltage. For instance, when a heavy electrical load is switched on, it experiences a high and extremely brief current, called current inrush, which induces a drop in network voltage [80]. This voltage drop (or voltage sag, or undervoltage) is felt by the entire network and thus affects the power available to the other systems, including the critical ones. This is illustrated in Figure 2, which exhibits the network voltage response of a 350 VDC test bench to a step consumption of a prototype electrically-powered Electrohydrostatic Actuator (EHA) [52, 127].

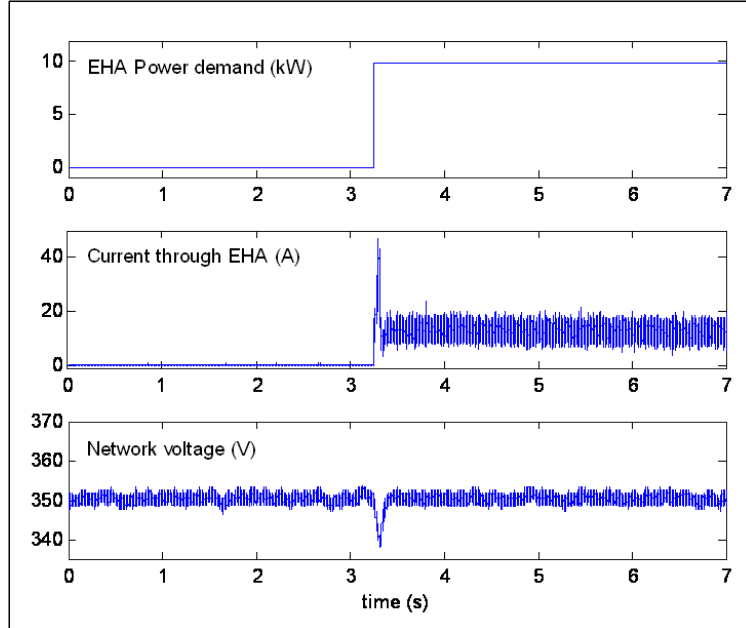


Figure 2: Power Quality Impact of Load Switching: Voltage Sag

Similarly, when a heavy load is switched off, for example in the case of a system failure, the electrical network experiences a short voltage increase (or voltage surge, or overvoltage), which can reach 600 V on a 115 VAC network for up to 1 ms [25]. These transients put an unacceptable level of stress on susceptible components causing loss of function and irreversible damage [19]. Such power quality issues pose

growing challenges in the aeronautical industry, with the current development of innovative “more-electric” architectures capable of meeting the ever increasing demand for electrical power [116]. Compared to traditional aircraft, these more-electric architectures incorporate more systems that are powered by the electrical network, which may potentially enable the downsizing (or even the elimination) of systems like the hydraulic and pneumatic networks. This is the direction taken by the “Power Optimised Aircraft” (POA) research project [3, 52]. Under the POA project, subsystems were more-electric, the engine bleed was reduced, and the electrical generators were directly mounted on the engine shafts, thus enabling the suppression of the engine accessory gearbox, as shown in Figure 3. These challenges are becoming ever more relevant since in these more-electric architectures, flight critical systems may draw power from the electrical network [65].

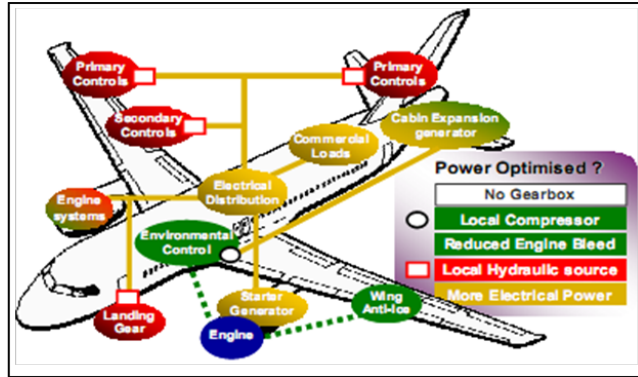


Figure 3: POA Architecture Concept [52]

From the examples above, it is now evident that transient regimes have potentially frequent and serious impacts on the operation of the aircraft and its subsystems. This important observation will be referred to as “Observation 1”:

Observation 1: Transient regimes experienced by dynamic systems can have potentially serious impacts on the operation of the aircraft.

Therefore, transient regimes must be closely monitored and controlled throughout the aircraft operation. Because of the inherent relationship between the operations and

the design of systems, transient regimes therefore intervene in the design of many systems. For example, Ganthony et al. explain that flight control actuators are sized according to the maximum transient load they may face during the mission [55]. These transient loads are very short and of a higher intensity compared to the steady-state current they normally draw from the electrical generator, as illustrated in Figure 4. Therefore, it is the transient regime operations, accountable for only a small portion of the mission, that drive the sizing of the actuator.

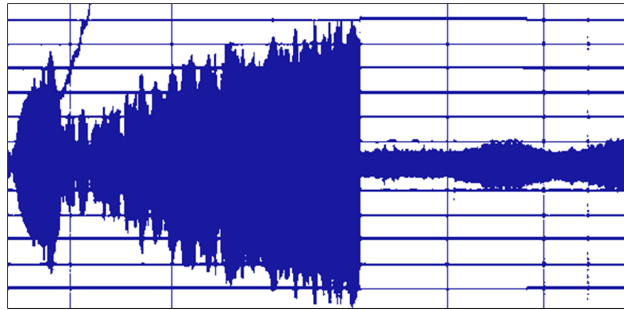


Figure 4: Transient Input Phase Current to an Aircraft Flight Control Surface Actuation System during a Typical Actuation Maneuver [55]

Similarly, an electrical generator is sized so that it can meet the maximum power required by transient peak demands of the electrical loads [1]. Also, some systems contain components whose sole purpose is to smooth the effects of transients, such as filters for electronic components. For hydraulic systems, accumulators and other control devices ensure that the system doesn’t experience drastic transient spikes in fluid pressure, as illustrated in Trikha [150]. These components, dedicated to transient regimes, thus increase the weight, the cost and the complexity of the overall system. This important role that transient regimes occupy in the design of dynamic systems is formally stated in “Observation 2”:

Observation 2: Transient regimes play a critical role in the design of dynamic systems.

1.3 *Transient Constraints in Design: Dynamic Constraints*

The design and development of systems are thus affected by the transient regimes the systems are expected to encounter. In particular, transient regimes sometimes induce a peculiar type of constraints on design responses: the value of these constraints vary with time, as illustrated in Figure 5. This figure depicts the transient constraint on the voltage of a 350 VDC network, as defined in the standards developed by the “Power Optimised Aircraft” research project [52]. From the instant at which the transient event is triggered ($t=0$), the voltage level has to stay within the bounds shown in the figure. One can see that depending on how long it has been since the transient regime was triggered by the perturbation, the maximum voltage allowed can vary from 427.8 V to 363 V.

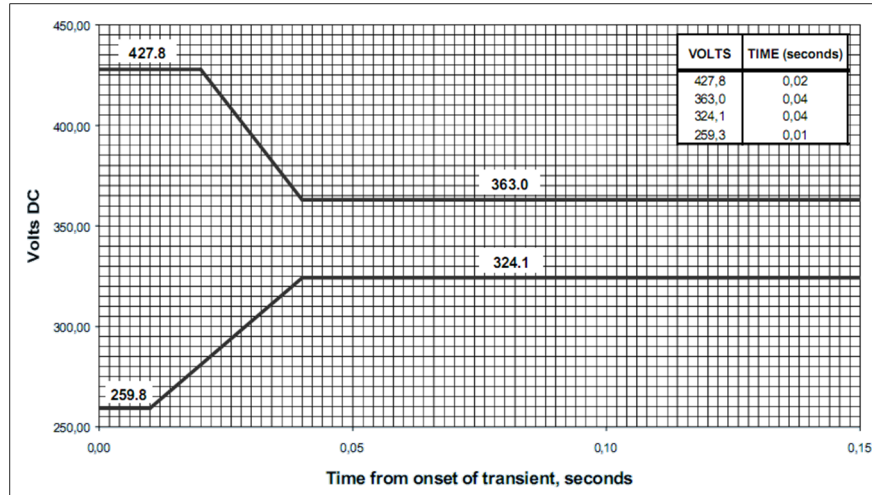


Figure 5: Transient Voltage Constraint for 350VDC Networks [52]

Dynamic constraints are not exclusively encountered with transient problem studies. An example of dynamic constraint can be found in the minimum time-to-climb problem for an aircraft evolving in a constrained airspace, where the allowable altitude boundaries, obviously tight in the vicinity of airports, vary with time along the mission [34].

As often with constraints, transient dynamic constraints can be subject to uncertainty. For instance, the voltage constraint of Figure 5 was applied by the POA program to a 350VDC network, for which no norm existed. It was thus derived from the existing power quality norms for 270VDC networks defined in the Military Standard 704 [41], and one of the research goals of the POA program was to study the pertinence of the new resulting 350VDC norm. The main point of this section can be summarized in the following formal observation:

Observation 3: In the design of dynamic systems, transient regimes are often addressed by means of dynamic constraints.

1.4 Aircraft Development Process

The previous paragraph showed that transient regime analysis was an influential part of the development of a dynamic system, and that it may intervene by means of dynamic constraints. Before discussing the specifics of the development process of a dynamic system and the challenges that arise from dynamic constraints, it is useful to comprehend the larger picture of the development process of the aircraft as a system.

1.4.1 Specification and Verification

Aerospace systems are complex by nature; a system is often composed of several subsystems, which are themselves assemblies of components and parts. The development of aerospace systems follows a cascade “top-down” process, where the design of higher-level systems induces requirements for the subsequent design of the subsystems, as shown in the traditional “waterfall diagram” represented in Figure 6 [36, 104]. The top-level requirements are mostly functional requirements, but as the system development advances through the lower stages, the requirements become technical specifications. One should note that in general, a system design activity (bloc at level N) will trigger several concurrent subsystem design activities (blocs at level N-1).

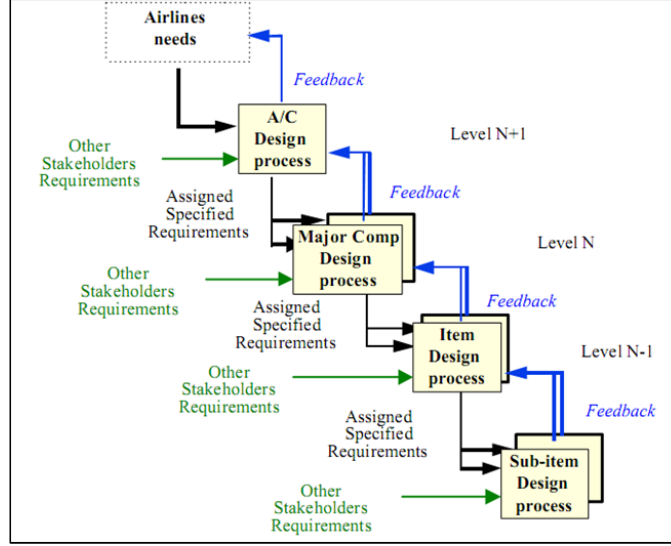


Figure 6: Aircraft Development Process: Waterfall View [36]

Each constitutive bloc of the waterfall diagram is itself a multi-step architecting process, as illustrated in Figure 7[86]. First, a design concept is selected. Based on the set of functional requirements received from the upper level, a choice is made on the components that will physically constitute the system under consideration. For instance, when designing an electrical circuit, one must first choose the components (resistors, inductors, capacitors, etc.). A first set of design parameters, characterizing the components, may be derived at the same time. The outcome of the concept selection activity is passed on to the design synthesis phase, where the design parameters of the system and its components are tuned. In the electrical circuit example, this corresponds to assigning values to the various resistances, inductances and capacitances [149]. This design synthesis phase often involves some optimization, where the outcome is a set of design parameters that minimizes or maximizes an objective function [151].

At the bottom of the waterfall (Figure 6), the aircraft and its systems are fully specified. But as one can see on the diagram, the process is not over, as there are feedbacks going from the bottom levels to the top ones. These feedbacks are the results of the verifications that occur at every step of the aircraft development

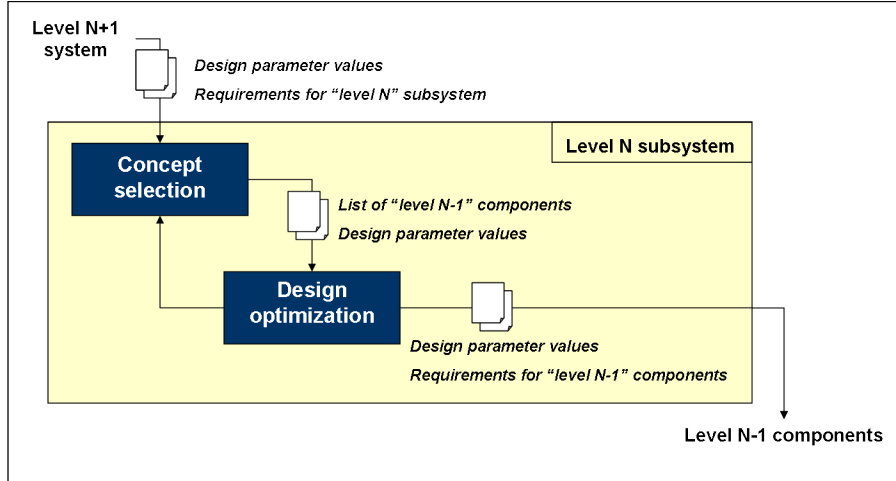


Figure 7: Design of Level N Subsystem

process. Indeed, after the specification of a subsystem, if requirements or constraints are violated, one course of action may be to go back up one or several levels, and go over the specification process again. This illustrates the highly iterative aspect of design. These feedback loops obviously conduce to inefficiencies of the design process, as they may induce rework costly rework and schedule delays.

One shortcoming of the waterfall representation of the aircraft development process is that it treats the feedbacks as merely a byproduct of design while they are in fact the output of the essential “verification and validation” activity. This shortcoming is addressed by another view of the development process, also widely adopted by the systems engineering community: the “V-diagram”, given in Figure 8[13, 2].

The V-diagram illustrates the role and interactions of the two major types of activity of the development process: specification and verification. Specification consists of making design choices, tuning the design parameters and deriving finer requirements to the lower-levels. It corresponds to going down the waterfall diagram. During verification, which goes up the waterfall diagram, the design is tested against the constraints and requirements.

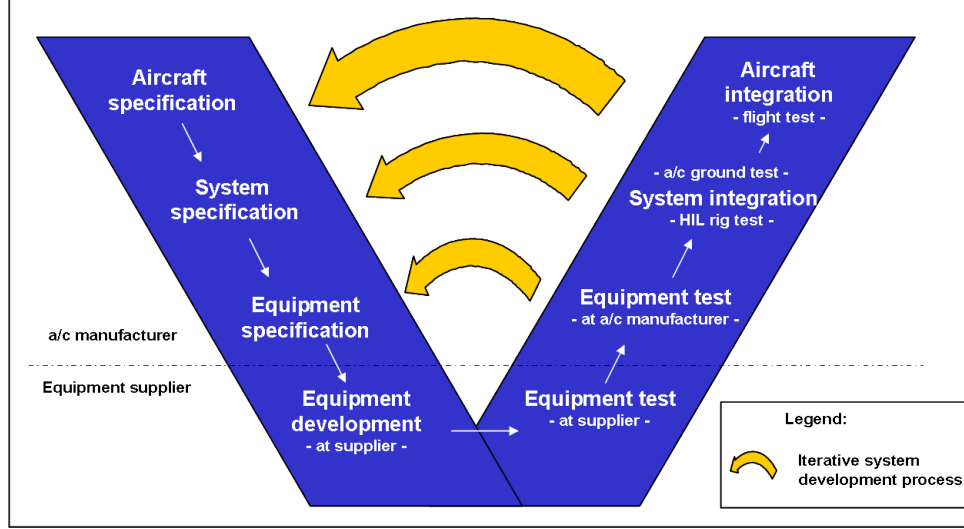


Figure 8: Aircraft Development Process: V-Diagram [13]

Before the advent and growth of computing capabilities, the bulk of verification and validation was done in the second branch of the V-diagram, where physical prototypes are assembled and integrated. For example, test benches incorporating the various flight control surfaces and their control systems, commonly referred to as “iron birds”, have become the norm within the aircraft manufacturing community for verification and certification [125]. Another example of test bench for the verification of integrated systems is the COPPER Bird[®] developed by Hispano-Suiza (Safran Group) in order to assess the behavior of aircraft electrical networks. Felix et al. describe the use of this test bench in the context of the POA program [52].

The verification process on a test rig follows stringent predefined procedures. For the POA program, the test scenarios as well as the procedures were defined in order to test the extreme cases. Test scenarios and procedures are often defined by regulations for the certification of aerospace systems. For instance, the testing procedures of 270VDC networks are documented in the corresponding “MIL-HDBK” guidance procedures [40].

If testing on the physical test rig exhibits failure of the system to meet requirements, the development of the system is delayed, the problem is investigated, and

eventually the design has to be revisited. The engineering work at this stage of the development thus involves an additional cost that can be high if a redesign is triggered.

1.4.2 Simulation: an Analysis Tool for Verification

With the development of computing capabilities, verification and validation now advantageously intervene earlier in the development process through *computer simulation*. As described in Law and Kelton, simulation consists in “imitating the operations of various kinds of real-world facilities or processes” [92]. In the context of aircraft system design, simulation consists in computing the behavior or the performance of the systems during their operations. The use of simulation can be represented in the V-Diagram by adding an intermediary “verification branch”, as shown in the modified V-Diagram of Figure 9 [13]. Simulation requires the creation of a mathematical model of the system, which is a “representation of the system in terms of logical and quantitative relationships” between input and output parameters [92]. This is illustrated in Kelemen and Imecs [81], where a substantive amount of effort is put into deriving the mathematical model of a motor system before simulating it.

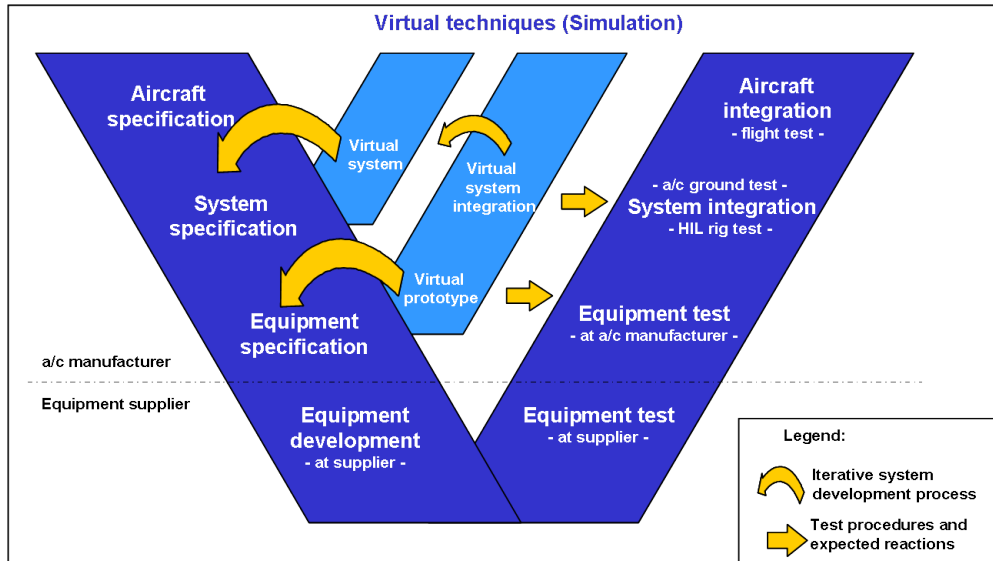


Figure 9: V-Diagram and Virtual Verification [13]

Simulation has gained wide acceptance and is now the norm, partly due to the development of software packages such as Simulink [106] or Dymola [45], which contain libraries that facilitate the creation of simulation models. In an example of the use of such software packages, an electrically-powered electro-hydrostatic actuator is modeled and simulated in Crowder and Maxwell in order to compute the time-behavior of motor velocity, fluid pressure and other actuator parameter [30]. The simulation helped the authors to conclude that their design has the capability of meeting the requirements of a power-by-wire technology.

On preliminary steps of the V-diagram, the lower-level subsystems are not fully designed yet, and assumptions on their behavior and performance are made, as illustrated in Woods, where a whole aircraft electrical system is modeled and simulated [158]. After each step of the specification branch of the V-diagram, more refined computer models of the systems are created and become more and more “physics-based”, as in the example by Crowder and Maxwell discussed above [30]. These refined models produce analyses with higher-fidelity, and correspond to the “Virtual prototype” label of the V-diagram of Figure 9.

As computing capabilities grew in power, it became possible to integrate the high-fidelity models of various subsystems together, thus producing multi-domain virtual test platforms that enable the system architects to analyze how the systems behave once put together [39]. For example, Chen et al. modeled and simulated switch-reluctance drives of aircraft electrical systems, and integrated the resulting detailed model to a model of turbofan engine [22]. In another example, Bals et al. integrated subsystem models into a “Virtual Iron Bird” in order to assess the performance of aircraft electrical architectures for more-electric aircraft [11]. This higher level of integration forms the core of the “virtual verification” and “virtual aircraft” paradigms, central to the Vivace research project [4], and illustrated by the modified V-diagram given in the labels “Virtual system integration” and “Virtual systems” in Figure 9.

1.4.3 Simulation as a Tool for Specification

Simulation became an essential part of the verification process. Moreover, with the improvement of computing capabilities, a new paradigm emerged in which simulation became a tool for the specification of systems. In this paradigm, several competing design concepts are analyzed through simulation and the design concept that exhibits better performance is selected for the next step of the development process. This elementary approach for the use of simulation as a design tool is illustrated in Figure 10.

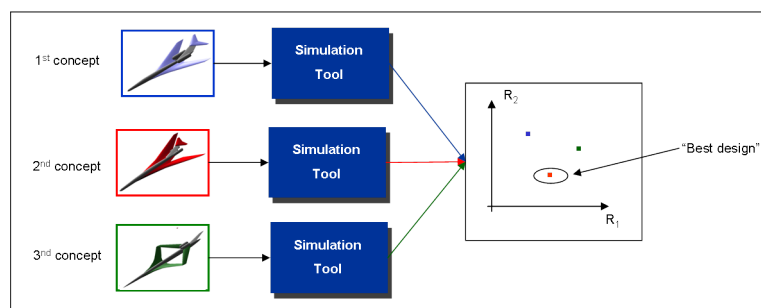


Figure 10: Simulation-based Design: Elementary Approach

By carrying more design choices longer in the development process, the design team gets better assurance that the selected design solution is the best. Obviously, the pertinence of this statement greatly depends on the ability of the design team to downselect the most promising candidate design solutions for analysis. It is also evident that the more design solutions are evaluated, the better confidence the design team will have on the final design concept choice. The ability of evaluating a high number of design candidates depends on the efficiency of the simulation tool. If the simulation tool is easy and efficient to run, then a vast number of design candidates may be analyzed. This forms the basis for the more advanced design techniques that are described in Chapter 2. On the other hand, if the simulation tool is heavy and time-consuming to run, then only a few design concepts can be evaluated and the

quality of the design relies more heavily on the expertise of the design team or on previous analyses.

1.4.4 Uncertainty in the Development Process

It was seen in previous sections that there can be uncertainty in the values of dynamic transient constraints. Uncertainty is a phenomenon that is inevitably present throughout the entire aircraft development process. In DeLaurentis and Mavris [38], uncertainty is formally defined as follows:

“Uncertainty is the incompleteness in knowledge (either in information or context), that causes model-based predictions to differ from reality in a manner described by some distribution function.”

As described in Dufresne [44], there exist different types of uncertainty (epistemic, aleatory), depending on the source and cause of uncertainty. In the aircraft development process, uncertainty can stem from changing requirements. For example, an aircraft may be designed with the objective of facilitating future upgrades, on which requirements are not clearly defined at the time of the design of the first version of the aircraft family, as illustrated in Lim [97]. Uncertainty can be induced by the fact that regulations may change, as was the case for the dynamic constraint on the 350VDC voltage of the POA program. One source of uncertainty is the fact that the design of a system at level N (cf. Figure 15) may necessitate making assumptions on characteristics and requirements for its subsystems (at level N-1 or lower), which are designed afterwards. Thus, these assumptions may be contradicted in subsequent design activities.

The potential feedback loops in the development process (cf. V-diagrams in Figures 8 and 9), induced by the verification activities, are often consequences of potential uncertainty. For instance, these feedback loops may be due to simulation errors introduced by mathematical assumptions in the modeling process. Another cause of

uncertainty that may lead to feedback loops is the lack of knowledge on the interrelationships of subsystems. For instance, in the POA program, the individual behaviors of system prototypes were well defined, but emerging behavior arose when the systems were integrated in the COPPER[®] Bird, which led to observed violations of transient constraints.

Because of the presence of uncertainty in the development process, traditional design methods that produce a single point solution become questionable. Instead, it is more advantageous to explore the design space, study the sensitivity of the design candidates to uncertainty effects, and carry families of solutions across design stages. Various advanced design methodologies have been developed in order to account for uncertainty, most of which are based on stochastic methods. Examples include the Robust Design Simulation process, described in Mavris et al., where a Monte-Carlo Simulation is performed on deterministic analysis tools in order to study the distributions and sensitivities of design responses with respect to design variables that are subject to uncertainty [107]. More details on the concepts of design space exploration, Monte-Carlo Simulation, and its applications to uncertainty analysis, are given in Chapter II.

It should be noted that systematic uncertainty analysis falls beyond the scope of this thesis. However, it is important to keep the issue in mind while investigating the efficient integration of transient constraints in the design of complex dynamic systems.

1.5 Dynamic Systems Development Process

The previous section gave an overview of the development process of the whole aircraft. It was seen that simulation now plays an essential role in the verification of system requirements and can also be used as a tool for the specification of systems.

This section focuses on the use of simulation in the development process of dynamic systems.

1.5.1 Time-Domain Simulation: First Introduction

In order to verify the good operation of a dynamic system, it is necessary to analyze the behavior over time of output signals of the system. Computing the behavior of signals of a dynamic system is the essence of Time-Domain Simulation¹ (TDS). As shown in Figure 11, TDS produces the time response of the dynamic system, given its design characteristics and a scenario under which it will operate. Thus, for a given system design, several operation scenarios can be investigated. Similarly, for a given operation scenario, several system designs can be evaluated.

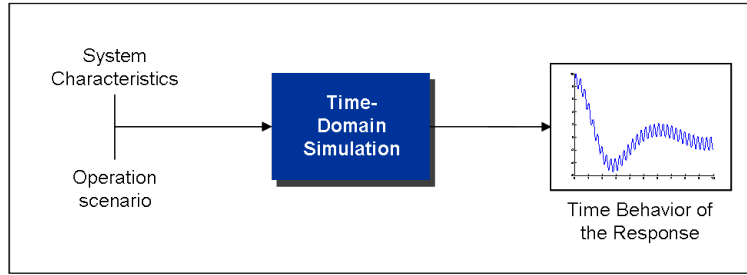


Figure 11: Time-Domain Simulation Process

Current TDS approaches require models incorporating detailed descriptions and behavioral laws of the components making the system. For example, in order to compute the voltage and current responses of an electrical circuit to a transient perturbation, one must know the nature of the components constituting the circuit (capacitors, inductors, resistors, diodes, etc.), and what the characteristic values of these components are (resistance, inductance, capacitance, diode threshold voltage, etc.) [60]. For example, in Arkada et al., the performance of two competing aircraft generator

¹For the sake of simplicity, “simulation” will hereafter refer to TDS.

designs is evaluated, after a complex and detailed modeling of the generators, taking into account the interactions between the armature windings, field, and damping circuits [7].

Because of the level of complexity that TDS models require, TDS typically intervenes in the later stages of the specification branch of the V-diagram (cf. Figure 8). And as will be discussed in the next section, TDS is usually used as a tool for analysis and virtual verification rather than for system specification.

1.5.2 Limitations of Time-Domain Simulation and Consequences

TDS has many limitations in terms of efficiency. Indeed, despite advances in computing power, simulation models can remain long to run for a variety of reasons. Firstly, encouraged by these new computing capabilities and by the need for a holistic approach to system design, models are becoming more and more integrated [10, 4]. Secondly, in a TDS model, each component, such as capacitor or inductance, is modeled via laws that govern the behavior of the dynamic responses. These laws are often formulated as differential equations. Thus, the simulation process often consists in performing successive resolutions of differential equations, using numerical solving methods [10]. The simulation time can thus increase exponentially as the simulated system gains in complexity [79].

Another factor influencing the simulation time of dynamic systems is the sample time interval of the simulation. This is the elementary time interval between two successive iterations of the numerical solver during the simulation. The need for better accuracy as well as the improvement in miniaturization has prompted the simulation community to perform simulation with smaller sample time intervals. This is especially true in the field of electronics and aircraft power systems, where growing use is made of power electronics systems with high-frequency switching semi-conductors.

These semi-conductors have switching frequencies that typically amount to hundreds of kilohertz [47, 115].

Therefore, TDS is primarily used as a verification tool instead of a specification tool, as it enables the design team to discover and analyze the behavior of the system. When used as a specification tool for design, the number of analysis runs is limited by the complexity of the model, and TDS is thus generally used as a “trial-and-error” tuning tool, in the fashion described in Figure 10. For instance, in Tatizawa et al., TDS was used to help the design of a high voltage transducer for DC measurements and power quality monitoring [143]. A few configurations, corresponding to different values of damping impedance, were tested with TDS, and the configuration exhibiting the better performance was selected. The efficiency of this process greatly depends on the expertise of the designers, who rely on rules-of-thumb or their experience during the downselecting of potential design concept solutions and the subsequent tuning of the design variables, as exemplified in Toumazou and Barry, where rules-of-thumb are derived in order to facilitate the design of analog circuits [149].

1.5.3 Controlled Dynamic Systems

As defined earlier, dynamic systems are those whose states are dynamic. In many instances, the dynamic system is required to behave a certain way in order for the mission to be executed well. For example, when the pilot gives the aileron a certain deflection command, the aileron deflection needs to attain the desired position within an acceptable amount of time and error tolerance in order for the aircraft to be maneuverable. With the high level of complexity in systems architectures, the physical links between the pilot and the end systems tend to disappear, as exemplified by the Fly-by-wire technology, whereby the pilot stick is linked with the flight control surface actuators through electrical signals [116]. Therefore, it became crucial to actively

control some systems at all times in order to make sure their dynamic responses reflect the input commands. These dynamic systems form a particular class of systems called “*controlled systems*”.

1.5.3.1 Description and classification

The control of dynamic systems is the subject of a vast field called “Control theory” [119]. Controlling the dynamic behavior of a system is generally done by adding another system (called “*controller*”) that feeds the controlled system (sometimes called the “*plant*”) with an input command. The combined subsystems that form the controlled system are generally designated by the term “*control system*”. There are two main types of control strategies: open-loop control and closed-loop control.

In open-loop control, the response of the plant to the commands given by the controller is well known. Therefore, no feedback on the current state of the plant is necessary [119]. The generic architecture of an open-loop control system is given in Figure 12.

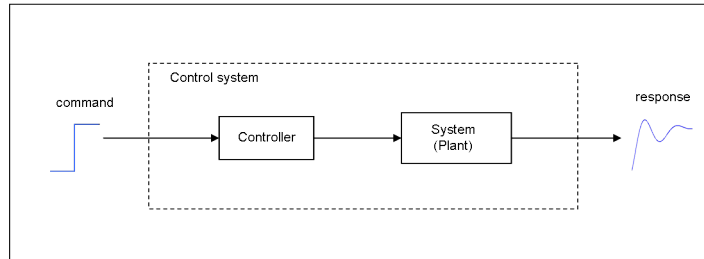


Figure 12: Open-loop Control System

In the case of closed-loop control, the command will be adjusted by the controller depending on the instant response of the controlled system. The latter thus needs to be monitored and fed back to the controller. This is the purpose (function) of a third kind of system (called “*sensor*”). The generic architecture of a control system with a feedback controlled loop is represented in Figure 13. It should be noted that the sensor is sometimes grouped with the plant to form a larger system called “*machine*”,

as in Ellis [46]. In other cases, the sensor is considered as exterior to the control system, as in Jeong et al. [70].

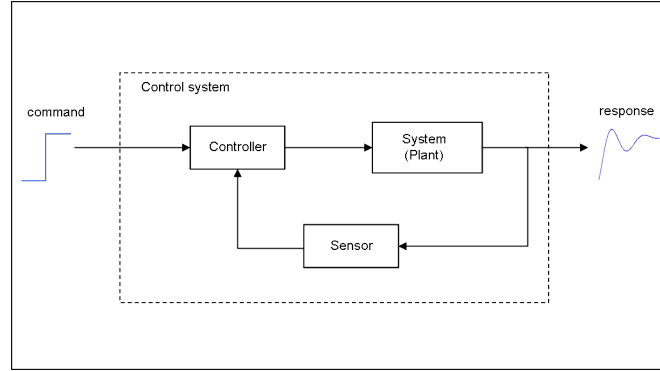


Figure 13: Closed-loop Control System

Most systems are however too difficult to predict to implement the open-loop strategy. This is because the controller usually incorporates a power converter that feeds power to the plant. The latter then uses this power and transforms it to produce the desired response. Because the behavior of the power converter may be affected by external perturbations, this additional level of power transformation generally requires instant monitoring of the system response and thus a feedback-loop control strategy [46].

1.5.3.2 Design of Controlled Systems

The design of a controlled system is a complex iterative process, as outlined in Nagrath [119]. Generally, the controller is designed after the design specification of the plant. Because the controller is a subsystem on its own, its design process can be related to the one outlined in Figure 7 (where requirements and design characteristics of higher-level systems trigger a series of design tasks). After a mathematical model of the plant is created, a controller concept is selected. Then begins a tuning process where the controller design parameters are adjusted. At this point, the control system is specified, and Time-Domain Simulation is executed to verify that the control system meets the requirements and constraints by checking if the dynamic response behaves

as desired. If this verification proves non satisfactory, then the design team has to go back to the fine-tuning process or, worse to the previous steps.

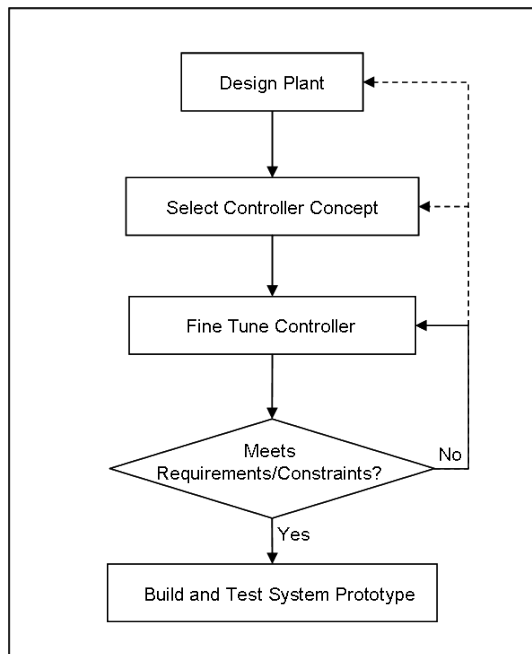


Figure 14: Controller Design Process

As explained in Ellis [46], tuning the control laws is often an “intimidating” process:

“often [tuning the control loops] is a combination of formal requirements and know-how gained with years of experience. Usually, only the most senior engineers in a company are capable of judging when a system is performing well enough.” [46]

Tuning the controller is thus a painstaking activity, that often relies on “trial-and-error” methods, as qualified by Ellis [46]. The induced inefficiencies result in long installation times for the controllers, from several days to weeks.

More modern techniques for tuning controllers involve formal optimization techniques. These techniques, which form the complex field of *Optimal Control Theory* [84, 68, 120], will be more thoroughly discussed in Chapter 2. Other potential approaches for reducing the inefficiencies of the design process of controllers are outlined

by Ellis [46]. They take advantage of the expertise of the control engineer earlier in the design process. This amounts to treating the controller and the plant as a single system and thus concurrently designing the two subsystems.

1.5.4 Dynamic Constraints: Challenges for the Design of Dynamic Systems

The design of a system often comprises an optimization process subjected to constraints. But as seen previously, in the case of dynamic systems, constraints may relate to their transient regimes and may therefore be dynamic. In order to verify that the proposed design solution satisfies a dynamic constraint, one must obtain the time evolution (or behavior) of the system, as it is generally not enough to ensure that the constraint is met at a limited number of time instants. Indeed, transient regimes often induce responses with fast dynamics.

Therefore, transient constraints that are dynamic require TDS or testing on a physical test bench. As seen previously, TDS presents limitations in terms of efficiency. Moreover, testing on a physical test bench generally takes even longer and is more costly. As a consequence, dynamic transient constraints are generally not fully taken into account during the design optimization, but in a subsequent verification activity (via TDS or physical testing), as illustrated in Figure 15.

If during this verification, TDS (or physical testing) shows that the proposed design does not meet the dynamic constraints, then design rework may be triggered. For example, in the case of an electrical circuit design, after the circuit components are selected and their design parameters are optimized, a TDS model is created and the circuit's behavior is simulated. If dynamic constraints are violated, the design team goes back to the optimization of the component parameters. This feedback process induces potential process inefficiencies and rework costs.

It was seen that at best, TDS can be used as a specification tool by analyzing a selected few candidate designs. Sometimes, the limitations of TDS make it so that

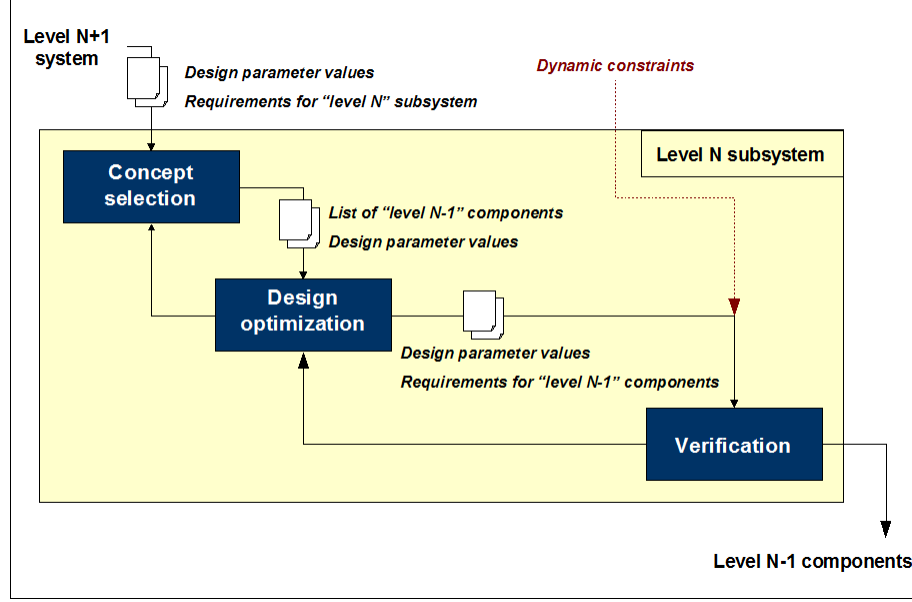


Figure 15: Design of Level N Subsystem with Dynamic Constraints

the transient regimes are not fully evaluated until the physical tests on the test bench. This was the case for the POA project, where the VIB simulation tool was used to design the more-electric architecture and the COPPER[®] Bird was used to validate the VIB [52, 11]. The complexity of the aircraft architecture combined with the limitations of TDS made it so that the VIB performed simulations of the electrical network at a sampling rate of 500 Hz, which was too low to capture the transient effects. Therefore, these were tested for on the physical test bench. Any design reiteration would therefore be costly. For instance, in Miri and Keyhani, the system under consideration, a power conditioning system aimed at improving the power quality of the electrical network, did not behave well enough in transient regimes. In order to improve the transient performance, one approach was to redesign the choke inductances and retest the improved system using TDS [112].

A more extreme example of design feedback due to dynamic transient constraints is the case of the International Space Station (ISS), exposed in Aintablian et al. [6]. Despite some use of TDS, testing on the test bench was the primary method of verification. This testing phase revealed that the design of the power system

did not meet the dynamic transient constraints related to power quality. Thus, the designers were left with the options of a costly redesigning the system or asking for an exception. The latter path was chosen, which required performing extensive analysis and testing to gain assurance that the design flaw was harmless. Therefore, the unforeseen additional engineering labor that was incurred had a negative impact on the cost of the development of the system.

The challenges induced by transient-related dynamic constraints can be summarized in “Observation 4”:

Observation 4: Dynamic constraints are generally not treated in the design optimization, but are dealt with subsequently during verification, using time-domain simulation (TDS) or physical testing on a test bench. This induces feedback loops in the development process, with potential for inefficiencies and rework costs.

1.6 Summary and Research Objectives

In the previous sections, observations were made on the importance and the role of transient regimes in the design and development of aircraft dynamic systems. These observations and first deductions can be summarized as follows:

Observation 1: Transient regimes experienced by dynamic systems can have potentially serious impacts on the operation of the aircraft.

Observation 2: Transient regimes play a critical role in the design of dynamic systems.

Observation 3: In the design of dynamic systems, transient regimes are often addressed by means of dynamic constraints.

Observation 4: Dynamic constraints are generally not treated in the design optimization, but are dealt with subsequently during verification, using time-domain simulation (TDS) or physical testing on a test bench. This induces feedback loops in the development process, with potential for inefficiencies and rework costs.

The main objective of this thesis is to develop a design methodology for aircraft dynamic systems, aiming at reducing the impact of feedback loops induced by the verification of transient related dynamic constraints. More specifically, the research strives to include these dynamic constraints in the design synthesis activity of Figure 15. This is summarized as follows:

Research Objective 1: Develop a methodology that integrates transient regime analysis and pertaining dynamic constraints into the design synthesis of aircraft dynamic systems.

This research objective will be the major driver behind all subsequent activities of this work. As the problem is refined and analyzed, it will be decomposed into intermediate objectives that are more manageable.

1.7 Organization of the Dissertation

Chapter I introduced the key concepts and gave the motivation for this thesis by identifying the need for a design methodology capable of capturing the verification of transient constraints early in the design of aircraft dynamic systems.

Chapter II delves deeper into the modeling, simulation and design of dynamic systems, describes modern optimization methods for design under transient constraints, and exposes the limitations of these methods before building the first elements of the design methodology that this thesis develops. Finally, chapter II exposes the need for the incorporation of a system identification approach into the design methodology.

Chapter III details the required characteristics of the system identification approaches to be selected. Then various candidate techniques are described before wavenets are eventually selected as the most promising approach. However, the potential shortcomings of wavenets are described, and in order to address them, an alternate approach is formulated based on the identification of the envelopes of the dynamic transient responses.

In chapter IV, methods for the extraction of signal envelopes are described, and a scheme for envelope identification of transient responses is formulated.

In chapter V, the hypotheses are summarized and the resulting design methodology is laid out in detail. Particular emphasis is put on the differences of implementation of the two competing approaches.

Chapter VI presents results of the implementation of the methodology, which is tested, along with its foundational hypotheses, on experiments that incrementally grow in complexity. First, the wavenet system identification approach is implemented on a conceptual dynamic system, represented by a simple mathematical law. Finally, the full methodology is implemented for the design of a notional 350 VDC electrical network.

Chapter II

ADVANCED DESIGN METHODS FOR DYNAMIC SYSTEMS AND FIRST HYPOTHESES

The previous chapter stated that the main objective of the proposed research is to develop a methodology aimed at improving the design process of aircraft dynamic systems by integrating the verification of transient dynamic constraints in the design optimization activity. This chapter explores the state of the art in the design of dynamic systems and explains in more detail the challenges that hinder the realization of the research objective. From these challenges arise research questions, which need to be investigated in order to fulfill the research objective. As an answer to the research questions, a first set of hypotheses is also formulated in this section. These hypotheses point to the foundations of a design methodology that will help attain the research objective.

2.1 Design Space Exploration of Dynamic Systems

2.1.1 Design Space and Operation Space

In the design of a system, a set of design variables is adjusted so that the design metrics (or responses) form a solution that is in accordance with the design requirements and constraints. A candidate solution is fully described by its design variables. It is referred to as a design case (or design point), and the set of all possible design cases constitute the design space. For example, the design space for a commercial aircraft includes all possible combinations of wing spans, wing areas, fuselage lengths, numbers of engines, tail configurations, etc. Designing a system can therefore be regarded as

selecting one point in a typically vast and highly multi-dimensional design space. In order to locate the “best” design point, one must gain knowledge on how the design metrics vary across the design space: this is design space exploration. This is the purpose of the study carried out in Briceno et al., where the design space for a supersonic business jet was explored [20]. Figure 16 shows a sample of wing planforms that were assessed during the design space exploration.

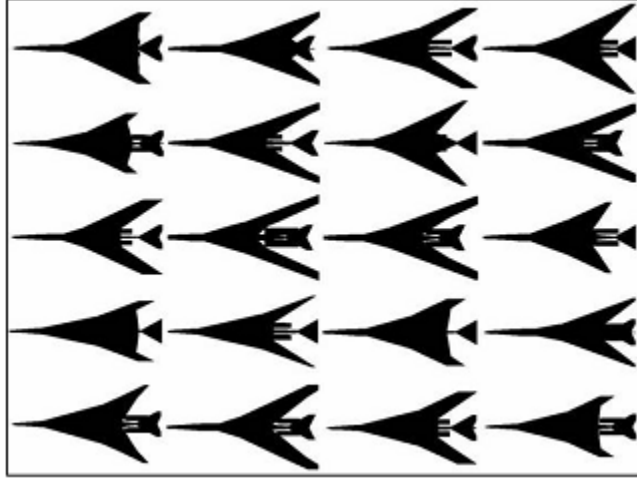


Figure 16: Design Space Exploration: Wing Planforms [20]

Similarly, verification can be viewed as the exploration of an operation space. Indeed, when verifying that a dynamic system behaves as desired, the system model or prototype is tested under several operating conditions. Each operating condition (or operation scenario) is described by a set of operation variables, which are generally included in the set of the system’s state variables. Examples of operation variables include the level of power that is drawn from an electrical generator, or the altitude of the aircraft.

The concepts of design space and operation space, and their relationship, are represented in Figure 17. In this example, the dynamic response of interest is computed for two specific operation conditions, for two candidate design cases. It must be noted that in the figure, the design and operation spaces are represented as cubes,

suggesting that all combinations of design variables and operation variables are feasible. This might not be the case, as incompatibilities might exist within the design space or within the operation space [21].

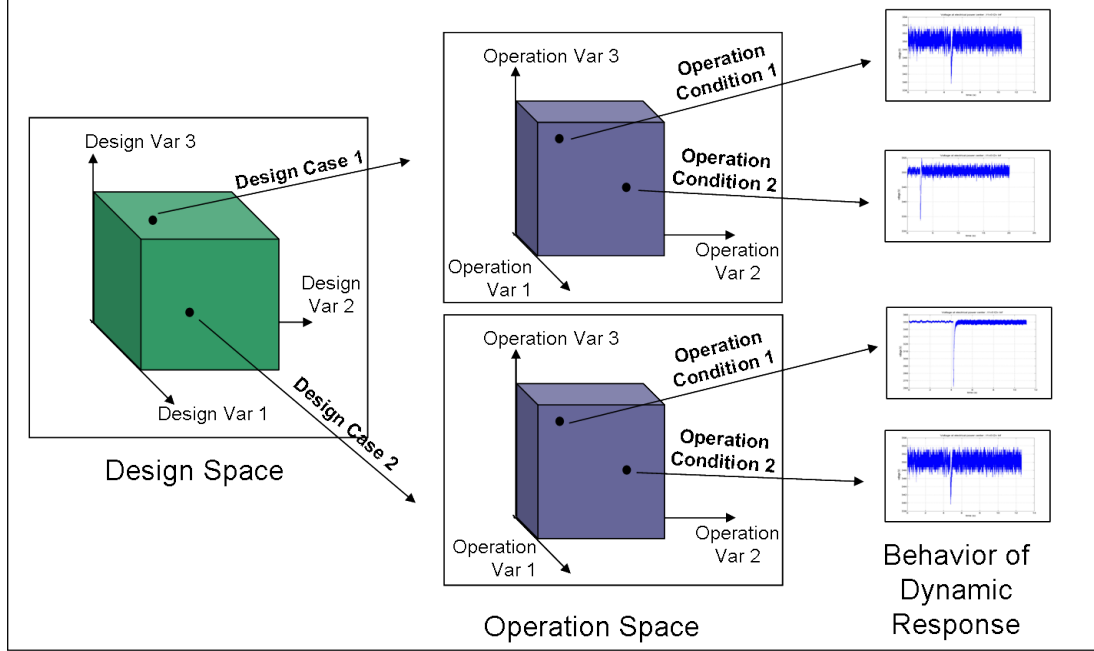


Figure 17: Design and Operation Spaces

Ideally, in the design of a system, all possible design points are evaluated (the design space is fully explored or scanned), and for each design point, all operation points are tested (the operation space is fully explored or scanned). Since many design and operation variables are continuous, this means analyzing an infinite number of design and operation points.

It was seen that the evaluation of transient dynamic constraints requires TDS, a single run of TDS corresponding to one design and operation points. Therefore, it is generally not possible to evaluate transient dynamic constraints for the entire design and operation spaces.

The design and operation spaces are therefore discretized. Still, even with a now finite number of design and operation points to evaluate, the size of the design and operation spaces impedes the thoroughness of their exploration. This was seen in the

first chapter with the example given in Tatizawa et al., where TDS was used to explore a few points of the design space of a high voltage transducer for DC measurements and power quality monitoring [143]. Similarly, when testing the transient behavior of a controlled system, the latter is verified on a limited number of points of the operation space.

The relationship between the design space and the operation space is an intricate one: as it was seen in the first chapter, the operation space has a direct impact on the design space exploration since systems are sized according to critical operation conditions. Therefore, the exploration of the design space must include the exploration of the operation space.

2.1.2 Optimization-based Methods

In the previous section, it was explained that performing a thorough design space exploration was a difficult task. To circumvent the obstacles against a thorough design space exploration, traditional design methods may use statistical regressions based on historical data. For instance, these statistical equations form core of the conceptual aircraft design methods found in Raymer. [129]. Simple examples of such statistical equations are given in Table 2, where statistical estimation equations of the thrust-to-weight ratio, defined as the ratio between the Sea Level Static (SLS) Thrust and Take-Off Gross Weight (TOGW), are based on existing aircraft.

Table 2: Thrust-to-Weight Ratio Estimation [129]

Aircraft type	Typically installed T/W
Jet trainer	0.4
Jet fighter (dogfighter)	0.9
Jet fighter (other)	0.6
Military cargo/bomber	0.25
Jet transport	0.25

The first shortfall of this approach is that it cannot be used for the design of systems that use revolutionary technologies, which bring a disruption to the past trends. For these, it is necessary to go back to fundamental models that are physics-based. Furthermore, this approach is clearly better-suited for the first stages of the conceptual design, where practical estimations of important metrics are sought, but it is not sufficient for dynamic systems subject to dynamic constraints, since, as seen previously, these require TDS (and thus a higher-fidelity model of the system) to be evaluated.

2.1.2.1 Optimization

The more sophisticated design methods rely upon the use of optimization algorithms, where design points are successively assessed, in an order determined by some logic. Optimization usually consists in determining a set of values for design variables that minimize (or maximize) an objective function (or a compound objective function). The problem at hand can be subject to constraints. A constrained optimization problem can be formalized in the following way:

$$\text{minimize: } f(\mathbf{x})$$

$$\text{subject to: } f_i(\mathbf{x}) \leq 0, i = 1..m$$

$$h_i(\mathbf{x}) = 0, i = 1..p$$

$$\text{where: } \mathbf{x} \in \mathbb{R}^n \text{ is the design variable}$$

$$f_i : \mathbb{R}^n \longrightarrow \mathbb{R}, i = 1..m \text{ are the inequality constraints}$$

$$h_i : \mathbb{R}^n \longrightarrow \mathbb{R}, i = 1..p \text{ are the equality constraints}$$

By nature, optimization minimizes or maximizes an objective function. However, most design problems are multiobjective: there may exist several concurrent responses

to minimize or maximize. For example, the design process of an aircraft tries to minimize the take-off gross weight while maximizing the dispatch reliability. In the case of the presence of multiple responses, these can be assigned weights and congregated into a single objective function, called Overall Evaluation Criterion (OEC). An example of OEC is given in Equation 1, where weights w_i are assigned to the responses R_i .

$$OEC = \sum_{i=1,n} w_i.R_i \quad (1)$$

Complex “real-world” optimization problems are solved using numerical optimization algorithms, which generally fall into either of two main categories: domain-spanning and path-searching optimization algorithms. For a good overview of optimization algorithms, the reader can study Van der Plaats [151], which served as the main source for the subsequent introduction paragraphs.

The mechanism of domain-spanning optimization is straight-forward: in a first step, a number of design points, spanning the design space, are each evaluated, and their performance is stored. In the next step, the design point that produced the better performance is chosen as the optimized solution. Path-searching optimization algorithms are more complex, and a generic form is given in Figure 18. The philosophy here can be viewed as advancing step by step closer to the optimal point. For example, in the case of the minimization of an objective function with two design variables, the goal can be viewed as finding the point on the corresponding surface with the lowest altitude. A path-searching optimization algorithm will start at an initial point, and will then go down the valley. At each step, the algorithm will use information from the previous steps or from the shape of the local area (by means of the gradient for example) to decide in which direction and by how much to advance. The search will end when a predefined convergence criterion is satisfied. For example, the algorithm

may consider that it has reached the lowest point in the valley when no significant change of altitude has been experienced for the last few steps.

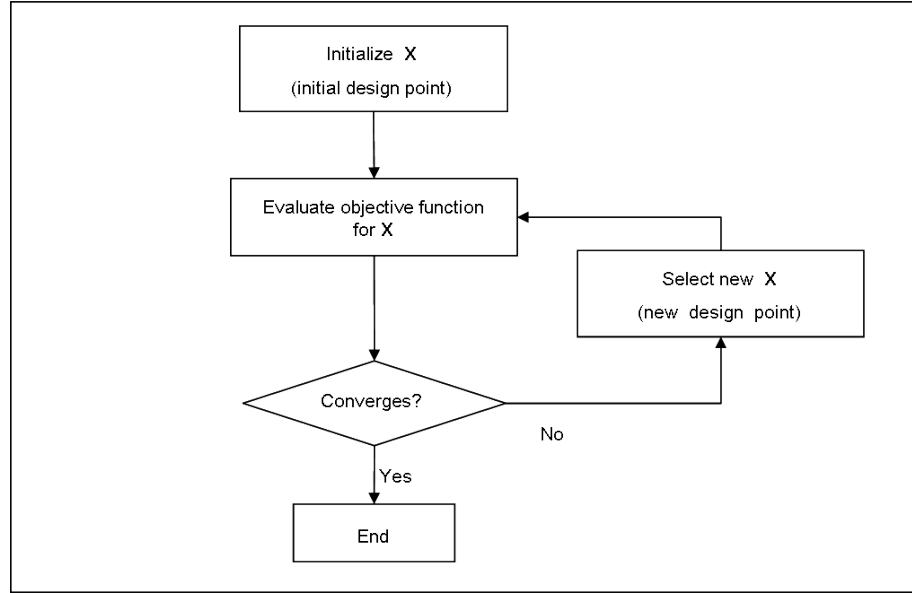


Figure 18: Generic Path-Searching Optimization Algorithm

In the presence of constraints, these are evaluated at each iteration of the optimization process (within the activity “Evaluate objective function” in Figure 18). In the previous example of the altitude minimization in a valley, there may be a constraint on the minimum altitude not to go under. In this case, at each step, the optimizer will evaluate if the altitude constraint is violated and by how much. There can be several strategies for ensuring that the final optimized point falls within the acceptable bounds. The optimizer may be forbidden to ever cross the altitude constraints. Or on the contrary, it may be allowed to do so, with a predefined margin, knowing that at some point of the process it will have to go uphill to reach an acceptable altitude.

Besides domain-spanning and path-searching optimization algorithms, there exist other forms of optimization, one of the most popular being genetic algorithms (GA). Genetic algorithms are based on Darwin’s principle of the survival of the fittest. In these optimization techniques, a pool of design points (population) are carried and modified across iterations. At each iteration, the performance (fitness) of every design

point of the population is evaluated, and a new population of design points (the next generation) is created such that characteristics of the better design points are favored [114].

Optimization is thus an iterative process, whether the chosen algorithm is domain-spanning, path-searching, or of another kind. When the system being optimized is subject to constraints, the latter are usually evaluated at each iteration, so that the outcome of the optimization is a candidate solution that satisfies these constraints.

2.1.2.2 Optimization for Dynamic Systems and Simulation-based Optimization

Optimization can be used advantageously in the design of dynamic systems. As one can see in the standard formulation of an optimization problem that was stated in the previous section, optimization is usually carried out on a static parameter $f(\mathbf{x})$. Therefore, when a dynamic system is optimized, static parameters describing the dynamic behavior (transient and steady-state) of the dynamic response are derived for optimization. Examples of typical transient-related parameters that are used for optimization are given in Kundur[91] and in Figure 19.

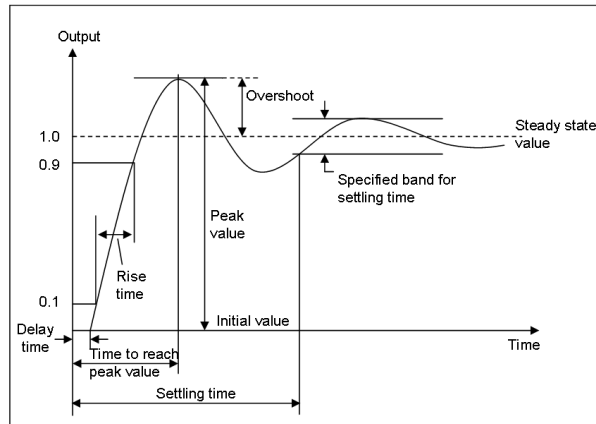


Figure 19: Transient Static Parameters for a Typical Response to a Step Input [91]

The parameters of Figure 19, which describe the transient behavior of the dynamic signal, may be subject to optimization when designing controllers of controlled dynamic systems. Optimizing the characteristics of a controller in order to assure

that controlled dynamic system behaves as desired is the subject of optimal control theory [84, 68, 120]. For example, in Crnosija et al., the speed response overshoot for a speed controller of a permanent magnet brushless DC motor drive is a response to be minimized [29].

In order to produce the transient static parameters, it is necessary to first compute the dynamic response, which is generally done through Time-Domain Simulation, as seen in the previous sections. Thus, at every evaluation of the objective function in an optimization activity, TDS is executed. The integration of TDS in the optimization process forms the essence of “simulation-based optimization”. Good review papers on the topic can be found in Fu [53, 54].

Examples of application of simulation-based optimization abound. For instance, in Jouannet et al., TDS was used to simulate the hydraulic flight control actuation system and to calculate the flight mechanics. The resulting TDS models were integrated within a more global sizing framework in order to determine the optimal geometrical configuration of an unmanned combat air vehicle [75]. The “Complex” optimization algorithm, derived from the popular “Simplex” algorithm, was used, during which the TDS models were run at each iteration.

Simulation-based optimization has been used effectively in the design of dynamic systems subject to transient-related constraints. For example, in Filizadeh et al., design characteristics of a DC current controller (composed of two subsystems) are optimized in order to minimize an OEC, sum of two individual objective functions. Each objective function is the integral squared deviation between the (dynamic) current response and the command for one of the subsystems[51]. The individual objective functions ObF_1 and ObF_2 are given by Equation 2.

$$ObF_i(x) = \int_{t=0}^{T_F} K(t) \cdot (I_d(t) - I_{ref}(t))^2 dt, \quad i = 1, 2 \quad (2)$$

The result of the optimization is a dynamic system that will see the current respond quickly to a command. However, because the transient regime has a shorter duration than the steady-state regime, the optimizer initially favored solutions that exhibited a dynamic response with large overshoots for extremely short durations. In their work, Filizadeh et al. thus included a weighting factor $K(t)$ that would “expand” the deviation of the current in the initial instants after the transient perturbation. The weighting factor, which forces the optimizer to focus on the transient regime, is represented in Figure 20.

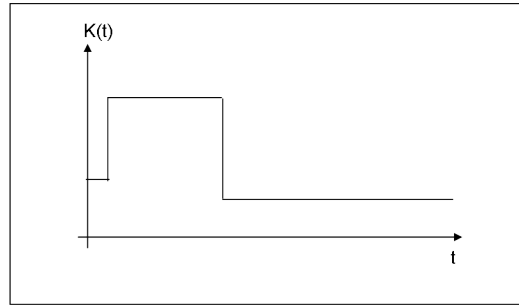


Figure 20: Weighting Factor $K(t)$ for Transient Regime [51]

At every iteration of the optimization, a new design point is produced for subsequent iteration. This new design point is evaluated by a TDS that was created for the simulation of transient perturbations. An overview of the optimization process is shown in Figure 21.

The transient optimization process used in Filizadeh et al. [51] is representative of a typical simulation-based optimization. The system under consideration in Filizadeh et al. was optimized to maximize the transient performance, but was not subject to the type of dynamic constraints discussed in Chapter 1. If the optimization had included dynamic constraints, these would have been evaluated at each iteration of the optimization.

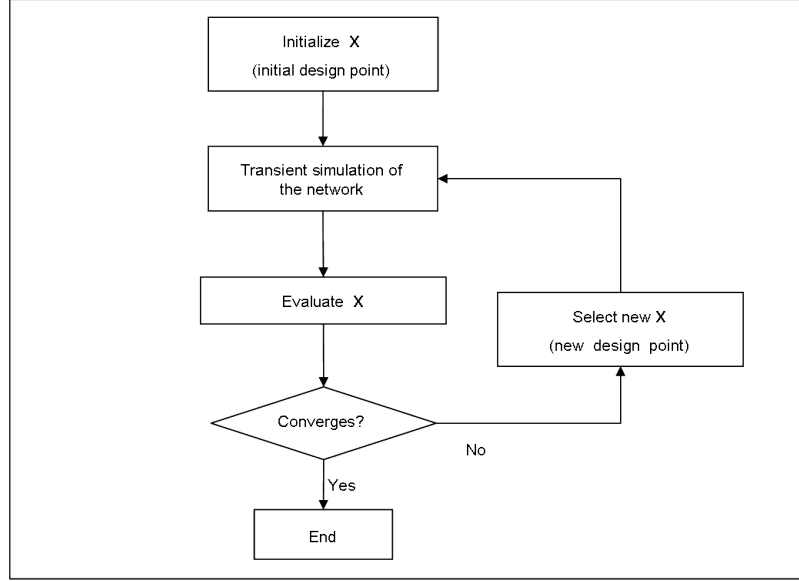


Figure 21: Simulation-Based Optimization for a Transient Controller [51]

2.1.2.3 Limitations of Optimization

The output of optimization is generally a unique solution that is carried on to the subsequent design phases. Conventional optimization thus perpetuates the paradigm of “point design”. In the context of the design of dynamic systems subject to dynamic transient constraints, conventional optimization methods, along with their point design philosophy, are victim to several shortcomings, which will be detailed hereafter.

Firstly, there are instances where the optimizer settles on a local optimum that is not a global one: it might be the best solution across a region, but not across the entire design space [151]. The concepts of global and local optimum are illustrated in Figure 22, where one can see the presence of a local minimum (a well) and that of a global one (the deeper well).

Secondly, having a unique solution makes it difficult to perform sensitivity analysis, which deals with studying the variability of the responses with respect to the design inputs. Sensitivity analysis thus gives a measure of the robustness of the solution, which is of great value since the design process is subject to many sources of uncertainty [44].

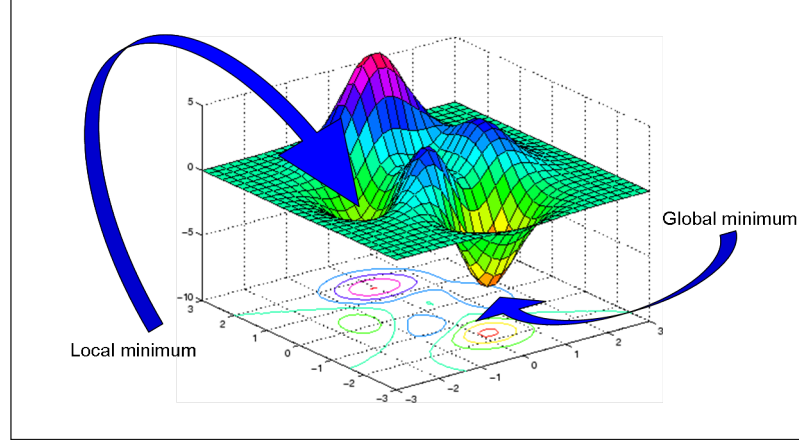


Figure 22: Local and Global Minima

Furthermore, in the design of a system, there are often several responses to monitor, minimize or maximize. Improving a response might be detrimental to another response. As explained in the previous paragraph, when an optimization algorithm is used, these multiple objectives are weighted and grouped into one single objective function, called the Overall Evaluation Criterion, which will be optimized [37]. The weighting assignment on the various responses implies the attribution of a hierarchy among them, which might introduce an improper bias towards particular responses. There might also be some uncertainty on the appropriate weightings, which increases the risk of later rework.

A major issue with the use of conventional optimization for the problem at hand is the fact that the constraints are “hard-coded” within the optimization algorithms. This constitutes a shortfall because constraints may be affected by uncertainty themselves, as discussed in Chapter 1. This makes it difficult to evaluate how robust the optimized solution is with respect to constraint uncertainty. Moreover, if the constraints are to be modified, one would have to redo the whole simulation-based optimization process from the start, an activity that the use of TDS makes slow. For instance, it was seen in Chapter 1 that in the context of the POA project, norms for

the innovative 350 VDC power networks did not exist, and the power quality constraints (cf. Figure 5) were derived by extrapolation from norms for 270 VDC power architectures [41]. Those newly-derived constraints may be modified when norms are subsequently set, which would require a complete and time-consuming repetition of the optimization process. This induces rework costs associated to the feedback described in the previous chapter.

The above-mentioned shortcomings are particularly relevant to path-searching optimization algorithms, since by nature, these provide a “best” solution without giving much insight on the shape of the global design space. The superiority of domain-spanning methods over path-searching is especially astute for the problems of local minimum avoidance and robustness to constraint uncertainty. Thus, domain-spanning optimization seems to be more appropriate for the problem at hand.

However, for most “real-world” problems, the high-number of design variables to adjust during optimization forms a design space that is often too large for a domain-spanning method to handle and span efficiently. This is especially true when dealing with dynamic transient constraints since these require time-consuming TDS in order to be taken into account.

In a nutshell, advanced design methods that rely on conventional optimization, perpetuate the paradigm of “point design” and fail to provide a means to efficiently explore multidimensional design and operation spaces while mitigating the effect of uncertainty. This leads to the formulation of the first research question, whose investigation will be crucial to achieve the research objective 1:

Research Question 1 (RQ1): How can one efficiently and thoroughly explore the design and operation spaces while accounting for dynamic responses and pertaining uncertain dynamic constraints?

2.1.3 Data Farming for Design Space Exploration

As explained in the previous section, traditionally favored design approaches stem from a point design philosophy. With the increase of computing capabilities, data farming, a more thorough method for design space exploration, has been developed. This method enables the designer to efficiently sample, visualize, and interactively query the design space through advanced visualization techniques.

2.1.3.1 Visual Analytics, Data Mining and Data Farming

This section provides a brief introduction to data mining, data farming, and visual analytics, three disciplines that are interconnected and that will be enablers for the efficient exploration of the design space of dynamic systems.

Data Mining: Discovering Knowledge in Data

With the coming of the age of information, the amount of collected and stored data has increased exponentially. A need arose to develop new tools that would enable to systematically explore the vast collection of data at hand. This is the purpose of *Data Mining*. As defined by Han and Kamber:

Data Mining “is the automated or convenient extraction of patterns representing knowledge implicitly stored or captured in large databases, data warehouses, the web, other massive information repositories, or data streams.” [61]

Data mining can be viewed as an activity that is encompassed by another one called *Knowledge Discovery in Databases* (KDD). As defined by Fayyad et al.:

“KDD refers to the overall process of discovering useful knowledge from data, and data mining refers to a particular step in this process. Data mining is the application of specific algorithms for extracting patterns from data.” [50]

Data mining has been used in a wide array of applications. For example, in Bhandari et al., a PC-based software is designed for the purpose of discovering patterns in the large statistic database of the National Basketball Association [16]. The resulting software, “Advanced Scout”, can detect anomalies and particularities, such as an unusually high field goal percentage for a player. The outcomes of the data mining process are presented as simple graphs or plain texts, which that can then be used by NBA coaches.

The example above demonstrates the use of data mining for discovering knowledge within “observational data”, which is passively collected (the basketball matches were not scheduled for the purpose of collecting statistics). Another instance of observational data is constituted by large databases collected in hospitals. These databases are formed by thousands of patients, treated over a number of years. It is obviously of vital interest for the biomedical and pharmaceutical communities to efficiently process these databases in order to discover interesting patterns and correlations. It is for the purpose of this kind of data mining activity that Cooper, in [26], presents a computer algorithm for the discovery of causal relationships in observational databases.

Visual Analytics: an Enabler for Data Mining

In the early 2000s, sectors like genomics (the study of the genomes of organisms) realized that they needed innovative tools to deal with the size, variety, and complexity of their data [157, 131]. A new discipline based on advanced visualization techniques and dubbed “*Visual analytics*”, started to grow, fostered by the growing capabilities of computing technology. As defined by Thomas et al.,

“Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces.” [146]

In a special issue of the “IEEE Computer Graphics and Applications” journal dedicated to the topic of visual analytics, the editors Wong and Thomas refine this definition:

“Visual analytics is the formation of abstract visual metaphors in combination with a human information discourse (interaction) that enables detection of the expected and discovery of the unexpected within massive, dynamically changing information spaces.” [157]

Visual analytics takes advantage of the high visual bandwidth of the human eye, colors, interaction, shape, etc. The principle of visual analytics lies in the use of advance visualization in order to foster the analysis of complex systems. Thus, visual analytics provides powerful techniques that push the barriers of data mining and KDD, by enabling the users to immerse in the data through their senses and interactively query it.

The popularity of visual analytics keeps growing, and numerous domains of application can be encountered. For instance, in the wake of the terrorist attacks of September 11th 2001, the United States Department of Homeland Security (DHS) increased the amount of surveillance data collected, and identified the need for new techniques for data mining. Visual analytics was identified as a potential solution, and the DHS established the National Visual Analytics Center [146].

In Wong et al., visual analytics techniques were used to visualize climate phenomena such as the typhoon that devastated the south of China in 1991 [156, 137]. The resulting maps were the fusions of multiple maps that focused on one parameter in particular. In the advanced maps, such as the overlay map shown in Figure 23, it became possible to comprehend several characteristics of the typhoon at the same time. In Figure 23, the colors and the streamlines represent the direction of the winds and the their structure respectively. The typhoon is the swirl seen in the bottom right corner. In another map, the wind speed was superimposed by means of relief shades. The aggregation of features into one advanced visual map is called data fusion. The data fusion techniques developed by Wong et al. will facilitate the visualization of the behavior of future typhoons and forecast their trajectories.

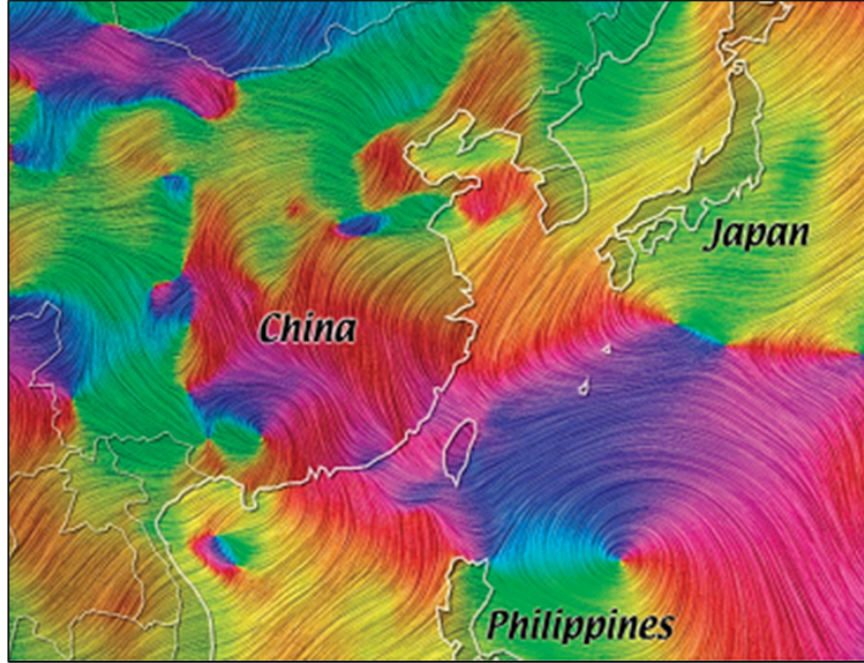


Figure 23: Visual Analytics for Typhoon Monitoring [156, 137]

As mentioned in the beginning of this section, genomics is a scientific domain that handles extremely large datasets. It has been found that in the human being, about twenty thousand genes express the proteins that are synthesized by the body. This amounts to billions of DNA base pairs [131]. Visual analytics has been a powerful enabler for the data mining of these genomics datasets. For instance, in the pharmaceutical industry, it is of great interest to pinpoint the genes that might be used as a drug’s “point of action”. Advanced visualization techniques described in Saffer et al. can help do that, for example by means of the heat map, shown in Figure 24 [131]. This heat map is similar to a spreadsheet that lists the response of the genes to different test conditions. Each row corresponds to a gene while each column corresponds to a test condition. The peculiarity of the heat map is that the response level is represented by a color instead of a number, which makes the spreadsheet more easy to comprehend. Test conditions (and genes) with similar characteristics can be clustered (grouped) to give the heat map more visible structure. The heat map is

interactive: when pointing at a colored region, the user can instantly query the name of the corresponding gene, test condition and response value.

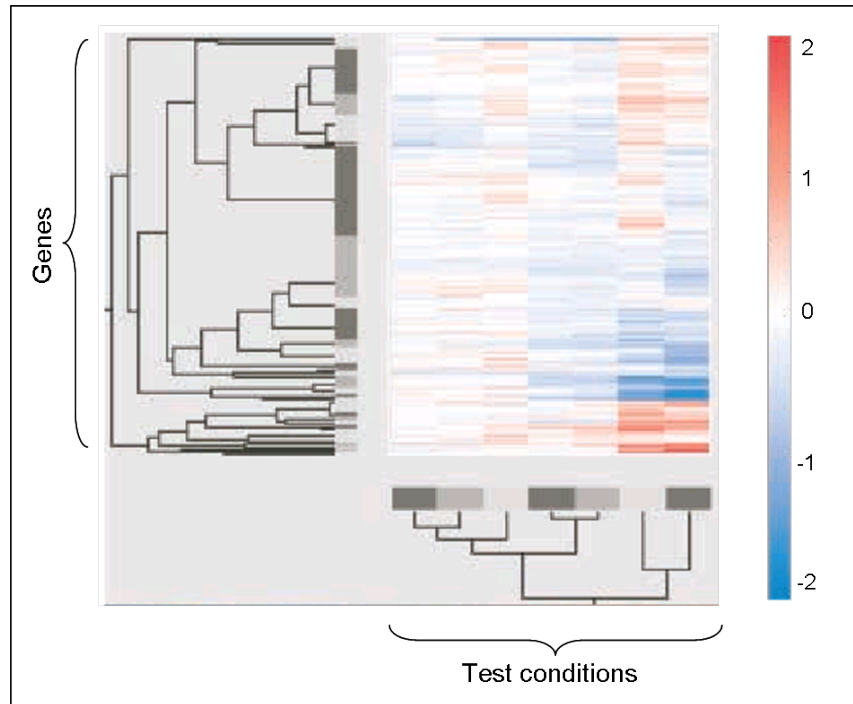


Figure 24: Heat Map for Genomics [131]

Saffer et al. lists another application of visual analytics in genomics, which is a visual correlation matrix [131]. A visual correlation matrix, shown in Figure 25, is used to identify patients who share similar characteristics. The correlation matrix shows the correlation with respect to 2856 genes for each pair of patients. The level of correlation is indicated by color (red for positive correlation and blue for negative correlation). Since each patient is correlated to themselves, the diagonal appears as a red line.

The human brain can visualize up to three spatial dimensions. Thus, many visual analytics techniques use three-dimensional plots to present the information in the data. For example, in Zudilova-Seinstra et al., streamlines in three dimensions were used to visualize the fluid flow streamlines computed by a Computational Fluid

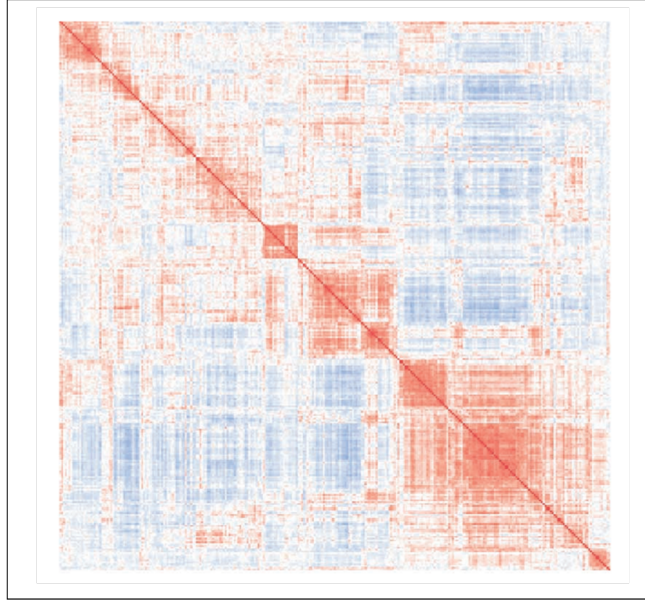


Figure 25: Correlation Matrix for Genome Matching [131]

Dynamics (CFD) model, as shown in Figure 26 [163]. Additional information such as speed can be overlaid using color schemes.

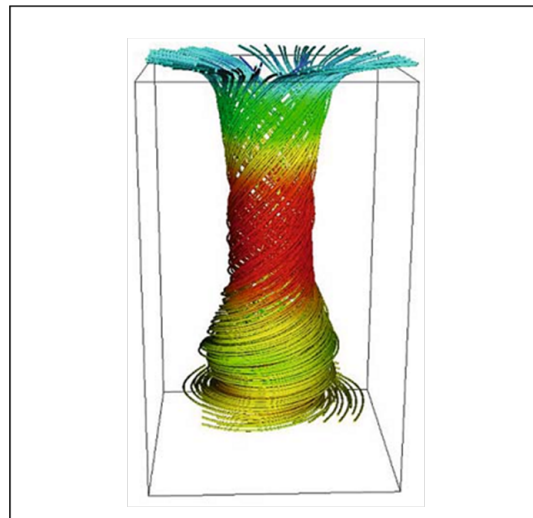


Figure 26: Three-dimensional Visualization of a CFD Model [163]

Visual analytics thus produces maps where multiple features are superimposed. However, the high complexity of the data often makes it impossible to collapse all the information into one single visualization map. In this case, multiple maps, showing different aspects of the multidimensional data, can be used concurrently and can be

linked so that user interaction on one map can be visualized on the other maps. An example is shown in Figure 27, where a heat map of the kind of Figure 24 (on the left side), exploring properties of various proteins, is linked to another visualization (the “galaxy view” on the right side) [131]. The proteins in the study have ligands (small compounds that bind to the proteins), and the complete library of ligands is visualized in the galaxy view, which plots them with respect to certain structural characteristics. When proteins are highlighted in the heat map (in yellow), the corresponding ligands are highlighted on the right side (the red points).

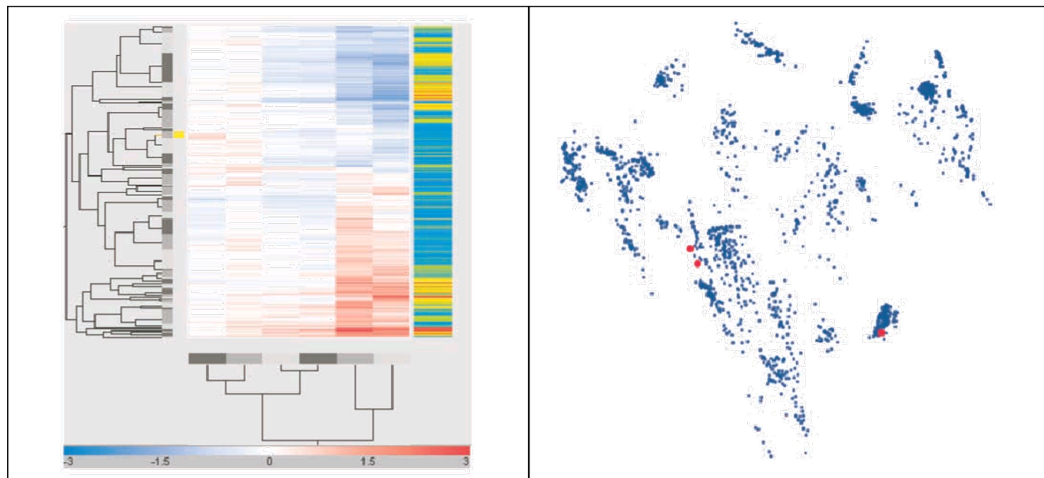


Figure 27: Linked Interactive Visualizations [131]

When the number and size of visualizations increase and when the amount of information overwhelms the screen, the data mining of the dataset may become tedious, which defeats the purpose of a visual analytics approach. Large displays, coupled with high resolution technology, offer a powerful remedy to this problem and enable the visualization of a large quantity of information at the same time [163, 153, 76]. An example of implementation of large displays for advanced visualization is given in Mavris et al., where the Collaborative Visualization Environment (CoVE), an advanced display wall at the Aerospace Systems Design Laboratory (Georgia Institute of Technology), is described (cf. Figure 28) [42].



Figure 28: Large Displays for Collaborative Visualization Environment [42]

Visual analytics thus facilitates the activity of data mining and knowledge discovering within existing data. It was seen for example that data mining is used to discover knowledge in the data accumulated by hospitals over years of service. Another kind of application of visual analytics for data mining is in the field of design. For example, the CoVE, which can expose the totality of the information related to a problem while typical displays typically represent about 15% of it, is linked to high-performance computing clusters. With this setting, multiple interactive parametric environments can be linked, so that the design team can, for example, make a change in the design parameter of a subsystem and study how the overall aircraft design changes [42].

Data Farming: Planting the Seeds for Design Space Exploration

In the previous sections, it was seen that data mining and visual analytics can enable the discovery of knowledge within large and complex datasets. In most of the examples cited above, data mining was carried on observational data that was collected without much control on the information. For example, a hospital has

no control on the type of patients that they treat. A hospital treats a localized area whose population has a limited genetic diversity. Thus, the hospital cannot rigorously derive knowledge about the entire humanity. For the purpose of design, it may be advantageous to have this control so that the data collected corresponds to the entire design space. Hence the more proactive concept of *data farming*. As described by Horne and Meyer, the main idea of data farming is to “grow more data in the areas of interest”[62]. A generic data farming process is illustrated in Figure 29.

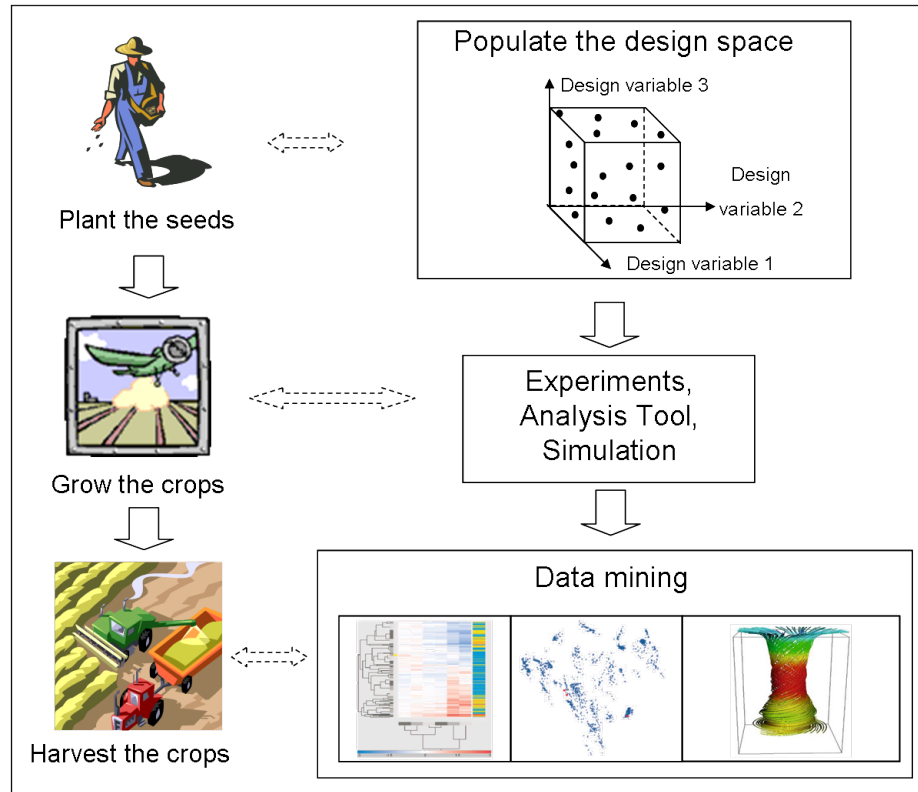


Figure 29: Data Farming Process

The defining element of data farming is the first step, which consists in populating the design space. This can be viewed as planting the seeds that will later produce interesting information for subsequent data mining. Ideally, the goal is to generate a vast number of design points that sample the entire design space, so that in the next steps, the extracted information allows for the thorough exploration of the design space. In the hospital example, this would be the equivalent of “growing” human

beings whose genes sample all the possible DNA combinations. Populating the design space for a thorough sampling may be done by means of techniques using random number generators, as in Ender [49]. This will be discussed more thoroughly in the next section.

After the seeds have been planted by populating the design space, the next step consists in “growing” data and information by running experiments, analysis tools, or simulation models. At the outcome of this step, the responses of interest for all the design points (seeds) are obtained.

The next step is data mining, where visual analytics techniques may be used to explore the design space, and perform trade-off studies. By interactively immersing within the data, the designer may play “what-if” analysis games and gain knowledge on the designed system.

After the data mining activity, it may be apparent that additional data is needed for some regions of the design space that are found to be interesting. In this case, the design team may wish to “zoom in”, and grow more data in this region, which amounts to performing another iteration of the process. This iterative approach is the one taken in Horne and Meyer, where the design team farms more data, depending on the results of the data mining activity [62].

In a nutshell, data farming leverages the capabilities of data mining by first fully seeding the design space. The resulting process provides a methodology that enables the thorough exploration of the design space. The next section will describe the implementation of a data farming approach in Ender[49].

2.1.3.2 A Data Farming Approach: Filtered Monte-Carlo for Static Parameters

Data farming has been used with success to efficiently explore the design space of systems with static responses. The Filtered Monte-Carlo technique, found in Ender

[49], is an example implementation of data farming for the exploration of a design space where static responses are subject to static constraints.

Populating the Design Space: Monte-Carlo Simulation

As suggested by the technique’s name, Filtered Monte-Carlo uses Monte-Carlo Simulation (MCS) to populate the design space and obtain the response data that is later investigated for thorough design space exploration.

The term “Monte-Carlo Method”, coined in the 1940s by physicists working in nuclear weapon projects [110], designates an array of methods based on statistical sampling, used to study a process that is a function of input variables. As illustrated in Figure 30, the essential principle consists of assigning distributions to the input design variables, using these distributions to statistically sample the design space thanks to a random number generator, and performing multiple runs on the Modeling and Simulation (M&S) tools (or on experimental workbench) such that the simulated design scenarios form a grid that samples the entire design space.

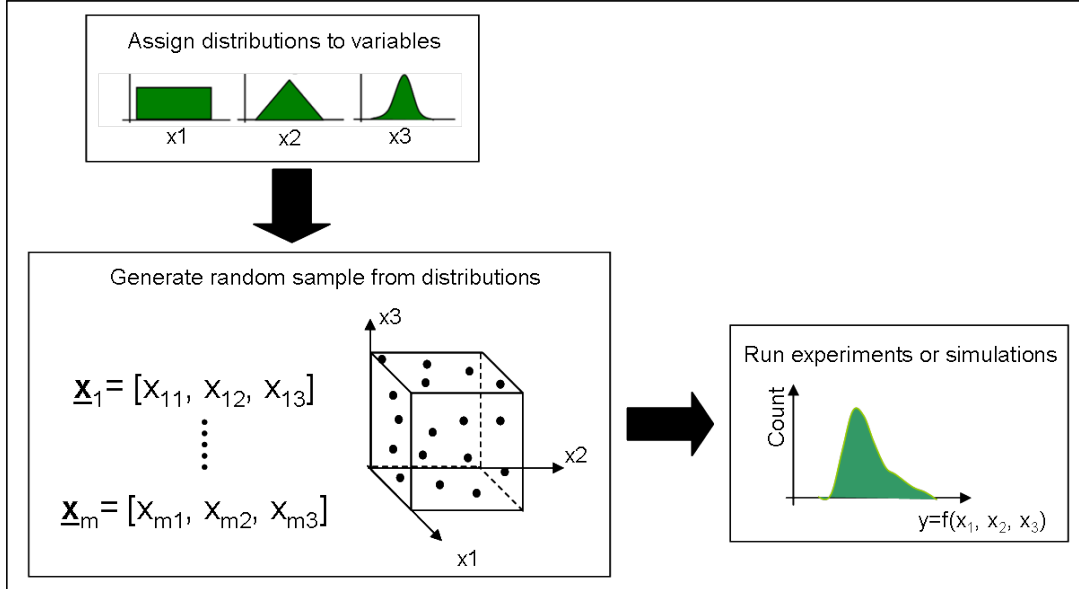


Figure 30: Monte-Carlo Simulation Process

The input distributions generally are uniform distributions. However, when one wishes to obtain more information on a particular region of the design or operation

space, other types of distributions, such as normal or triangular distributions, may be chosen to introduce a bias towards that region.

Monte-Carlo simulation has been widely used for uncertainty analysis. Most models used in design are deterministic in nature: to a set of input values corresponds a unique set of responses. However, most processes are subject to uncertainty. The reasons of this uncertainty are numerous. For example, there may be noise induced by uncontrollable parameters or hard assumptions that were made when creating the design tools. A Monte-Carlo simulation on the deterministic design tools can be used to model the uncertainty of the design outcome by assigning distributions to noise parameters. For example, in Mavris et al., Monte-Carlo simulation was used to analyze the uncertainty in the design of a High-Speed Civil Transport and to produce a “robust” solution that would be less sensitive to potential variations of noise parameters (such as fuel price) [107].

Within the Filtered Monte-Carlo technique, the outcome of Monte-Carlos simulation is a set of responses that corresponds to design points that span the entire design space. Thus, this activity encompasses the first two main activities of the data farming process illustrated in Figure 29.

Visual interactive environment for design space exploration

After the responses are obtained by Monte-Carlo Simulation, the next step of the data farming approach is to perform data mining in order to obtain knowledge on the design space. In the Filtered Monte-Carlo technique, the knowledge harvested pertains to how the responses of interest vary across the design space. In the end, the designer can instantly select the design points that match design criteria.

The simulated design points, along with their responses generated by Monte-Carlo simulation, are imported into an interactive visualization environment [49]. The purpose of this technique is to allow the designer to instantly visualize the entire design space, and determine regions that verify design constraints or other regions of

interest. The design points and their response values are visualized in clusters within a multivariate scatterplot matrix, as shown in Figure 31.

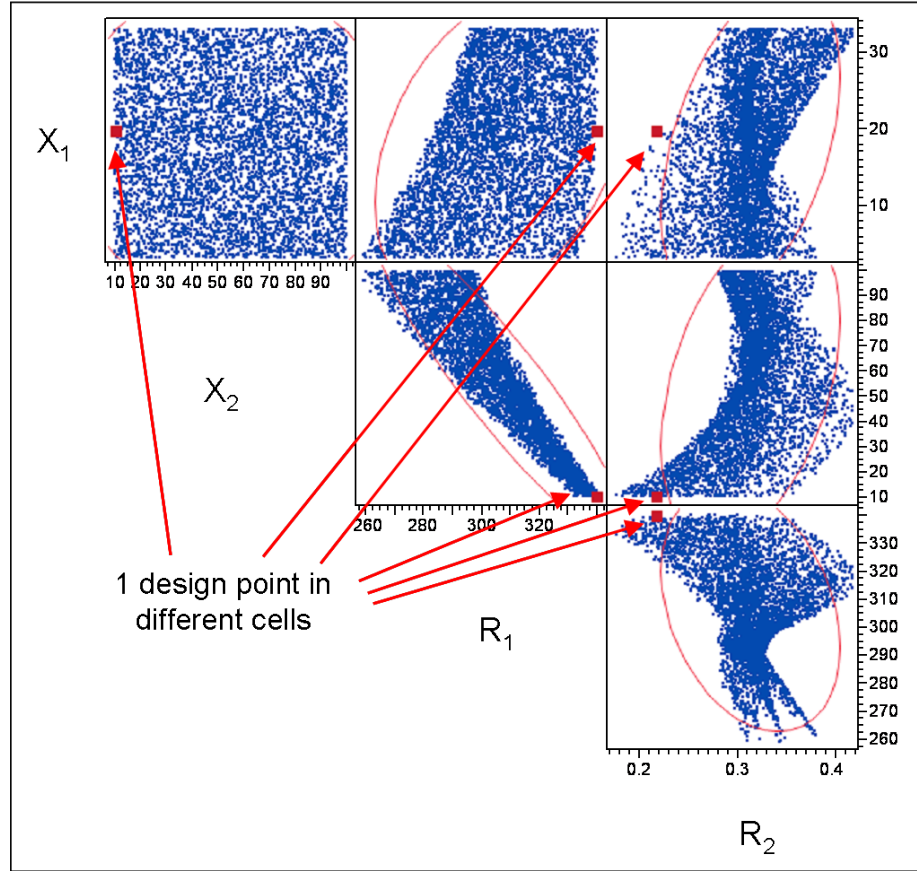


Figure 31: Scatterplot Matrix for the Filtered Monte-Carlo Technique

The scatterplot matrix is one of the oldest, simplest, and still most popular visualization techniques for multidimensional data [48]. It essentially consists of an aggregation of plots of pairwise dimensions. In the scatterplot matrix, each row or column represents a parameter, whether it is an input parameter or a response, and these parameters are plotted against one another, thus forming cells. The example shown in Figure 31 is representative of a design problem with two design variables (X_1 and X_2) and two design responses (R_1 and R_2). Each point within a cell corresponds to an operating/design point, i.e. to a setting of the input parameters. One can see on the figure that in the cell plotting X_1 against X_2 , the entire space is filled, which

indicates that all the design regions are explored. It is worth noting that the usual representation of the scatterplot matrix is the square scatterplot, which is symmetrical. The scatterplot shown in Figure 31 represents the upper triangle of the square matrix. Obviously, each cell of the diagonal line of the square matrix (plotting X_1 vs. X_1 , X_2 vs. X_2 , etc.) is filled with points that exclusively fall on the $y = x$ identity line.

In the Filtered Monte-Carlo of Ender [49], the environment is interactive and the cells are linked to one another, so that any action of the user in one cell is instantly propagated to the other cells. Thus, when the user selects a point in a cell, the environment indicates the location of the corresponding design point in the other cells.

Another feature of the visualization environment is the ability for the user to add constraints on the parameters and instantly filter-out the design points that meet (or do not meet) the constraints, as illustrated in Figures 32 and 33. Feasibility regions can thus be highlighted and selected. The interactivity of the environment enables quick investigation of several constraint scenarios and the visualization of how the design options are sensitive to the uncertainty on the constraints.

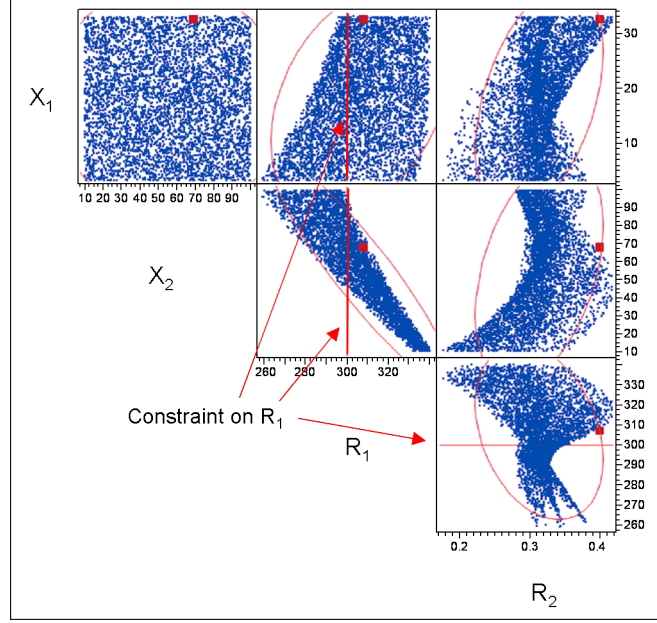


Figure 32: Scatterplot Matrix with Constraints for the Filtered Monte-Carlo Technique

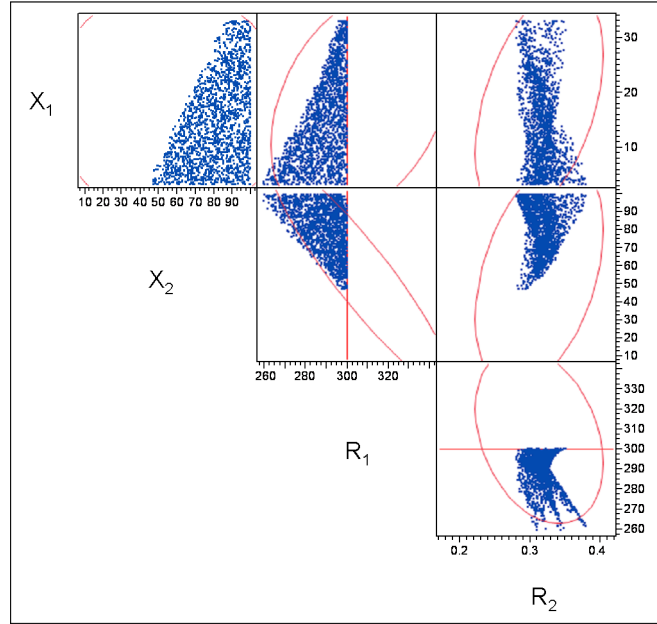


Figure 33: Scatterplot Matrix after Filtering for the Filtered Monte-Carlo Technique

So far, the Filtered-Monte-Carlo technique has been applied to design problems with static metrics. In Ender [49], it was applied to efficiently investigate the complex design space of a system-of-system problem with multiple levels of responses. These

levels corresponded to the position of a subsystem in the overall system hierarchy. The metrics considered were purely design metrics, related to the characteristics and capabilities of the system. In Phan et al. [127], the Filtered-Monte-Carlo technique was used to visualize the operational behavior of an electrical test rig for electrical signals. These signals were, by definition, dynamic; however, the responses used in this study were static metrics derived from the signals, such as voltage amplitude or voltage peak level. It was shown that for these static metrics, the Filtered-Monte-Carlo technique enabled the efficient and thorough exploration of the operational behavior of the electrical test rig.

The capabilities of the Filtered-Monte-Carlo can be succinctly summarized in the following observation:

Observation 5: The Filtered-Monte-Carlo technique enables the efficient and thorough exploration of the design/operation space composed of static metrics.

2.1.4 An Innovative Data Farming Approach for the Design of Dynamic Systems: Time-Domain Filtered Monte-Carlo

Because the notion of time does not exist in the multivariate scatterplot of the Filtered-Monte-Carlo, the technique cannot be readily applied to problems dealing with dynamic signals. However, the promises exhibited above lead to hypothesize that the technique may serve as a basis for a new methodology that allows the efficient and thorough exploration of design and operation spaces with dynamic signals and dynamic constraints. This can be reformulated as follows:

Hypothesis 1 (H1): A data farming approach, based on the integration of time as a dimension in the Filtered-Monte-Carlo approach, will conduce to the efficient and thorough exploration of the design/operation space for dynamic signals with dynamic constraints.

This extended Filtered-Monte-Carlo technique, which will be called hereafter Time-Domain Filtered-Monte-Carlo, would thus help fulfill the research objectives and answer the research question formulated above.

The essential principle of the Time-Domain Filtered-Monte-Carlo remains the same as that of the Filtered-Monte-Carlo: populating the design space and growing the data are done running a Monte-Carlo Simulation. However, at this stage, the data to be explored not only consists of static responses, but of dynamic signals. Therefore, the main idea to be investigated here consists in adding a new type of elementary cell to the multivariate scatterplot matrix of the Filtered-Monte-Carlo technique. While in the original technique, the elementary cells are all bivariate plots of the static metrics, the new cells will visualize the massive datasets representing the time-behavior of the dynamic signals obtained after the Monte-Carlo Simulation. The design of the new time-domain cells will be discussed in the next section. Because the objective of the Time-Domain Filtered Monte-Carlo is to interactively explore the design space of dynamic systems, the components of the visual environment (the scatterplot matrix and the time-domain cells) will be automatically linked. The first vision of the structure of the visual environment for the implementation of the Time-Domain Filtered-Monte-Carlo is shown in Figure 34.

2.1.4.1 Data Mining and Visualization of Time-Domain Signals

There are many ways of visually representing massive datasets of dynamic responses and performing visual data mining on signals. In order to choose the most appropriate one for the implementation of the Time-Domain Filtered Monte-Carlo, a brief survey of visual data mining on time-domain signals is presented in this paragraph, based on a few examples found in literature and on the more thorough overview given in Lin et al [98].

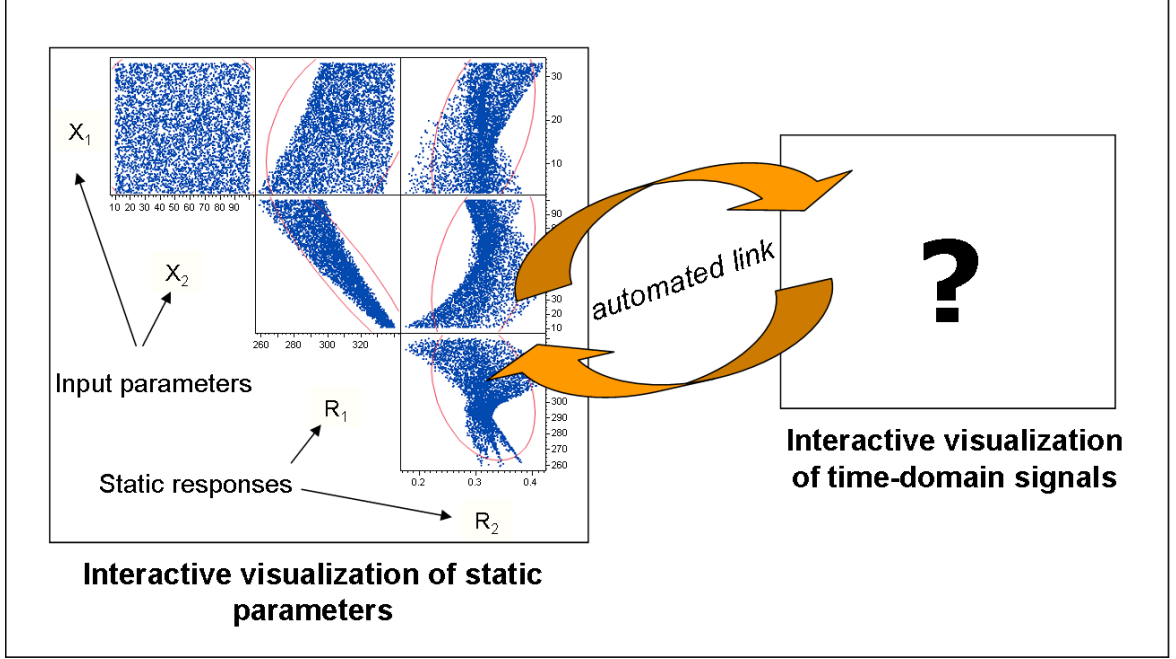


Figure 34: Time-Domain Filtered-Monte-Carlo: First Description of the Visual Environment

One way to visualize a collection of dynamic signals is to plot them in three dimensions. The axes correspond to time, signal number, and response value. An example, shown in Figure 35, can be found in Wijk and Van Selow, where each signal represents the hourly electrical power consumption, over a day, of a research center [155]. Depth is added to the plot by adding an axis corresponding to the day of the year. The signals are then placed side by side so that the power consumption over a year can be visualized.

The visualization shown in Figure 35 enables a global understanding of the behavior of the signals. However, it can be difficult to investigate some regions that may be obstructed by surrounding regions. For example, in Figure 35, the flat consumption levels of week-end days are hidden between the crests corresponding to week days. A solution for this problem is proposed by the “clustering and calendar-based” approach, described in Wijk and Van Selow [155]. The first step of this technique consists in “clustering” the data, a very common technique for data mining, where the elements

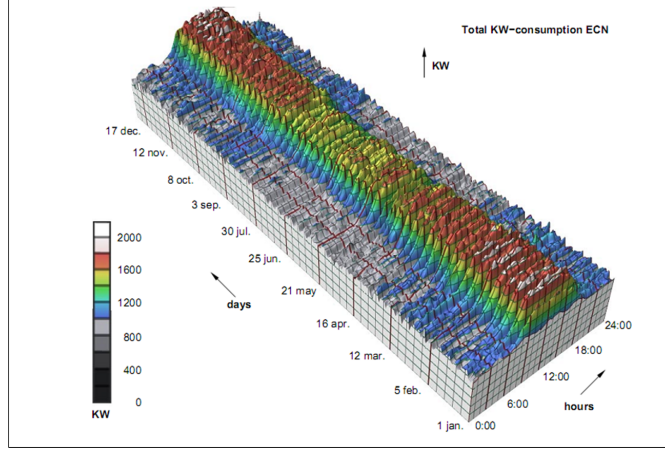


Figure 35: Three Dimensional Visual Representation of Power Consumption [155]

of the dataset are grouped into homogeneous clusters according to specific features. Clustering techniques for time-domain signals are numerous and surveyed in Liao [95]. In the example of the power consumption monitoring above, a few typical power consumption profiles are identified and taken as the bases of the categories. Then, every power profile is assigned to the category of the base profile most similar to it. Finally, the time-domain signals are viewed in two dimensions using the calendar based approach, where the days of the year are marked by color depending on the cluster they belong to (cf. Figure 36). For instance, the flat consumption lines are grouped into one cluster, represented by the flat blue line. The corresponding days, which are mostly week-end days, are highlighted in the calendar.

In some cases, it may be desirable to be able to visualize all the time-domain signals, without (or before) clustering. This is true for the implementation of the Time-Domain Filtered-Monte-Carlo, where the user may need to isolate signals during the design space exploration. A way to visualize sets of signals is simply to aggregate them in a single bi-dimensional “overlay” plot. Figure 37 gives an example of such an overlay plot developed for “TimeSearcher”, a visualization tool for dynamic data [155].

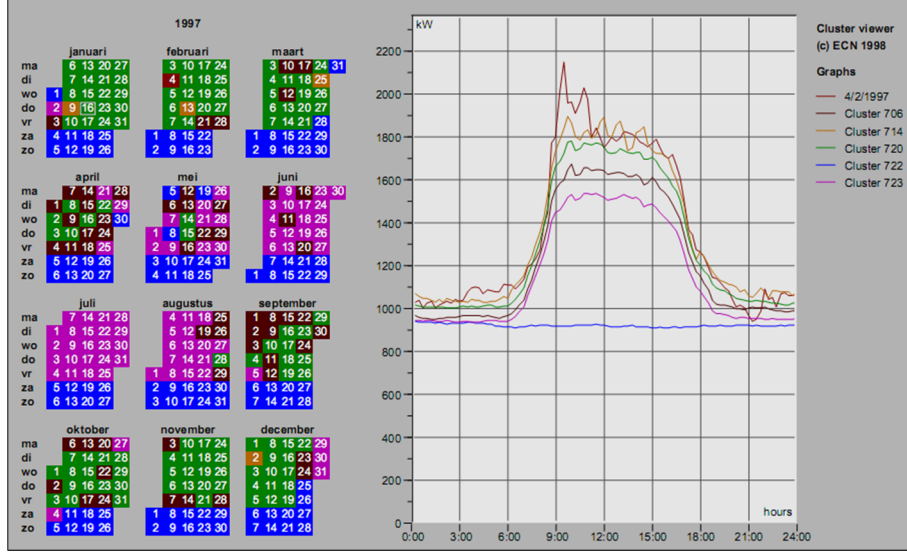


Figure 36: Clustering and Calendar Based Representation of Signals [155]

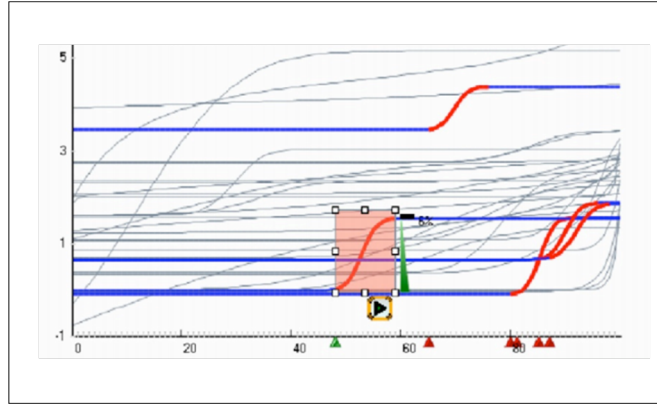


Figure 37: TimeSearcher Visual Interface

In order to avoid the difficulties arising from clutter and be able to make sense of the data, the overlay plot must be interactive. In TimeSearcher, the user can zoom in and out within the data. The user can also specify a shape of interest (the red box in the figure), and the data mining tool then finds the signals of the dataset that contain the desired pattern.

When the signals of the dataset contain periodicity information, they can be visualized using the “Spiral Visualization” technique defined in Weber et al [154]. This technique was used in Lin et al. to visualize the dataset of the power consumption

example described above [98]. In the resulting graph, shown in Figure 38, the periodic structure of the data becomes apparent.

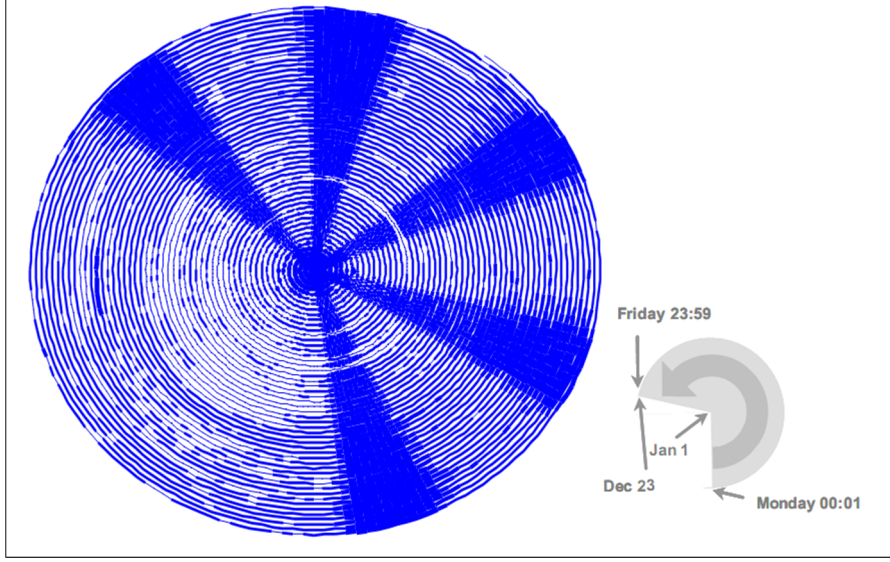


Figure 38: Spiral Visualization of Power Consumption [154, 98]

As one can see from these examples, there are various ways of representing time and time-domain datasets. The design of a visualization environment for the Time-Domain Filtered-Monte-Carlo developed in this thesis is therefore a task subject to several design criteria. These criteria are formalized in Aigner et al., where the main features and design options for such a visual environment is exposed [5]. The resulting table is presented in Table 3.

Table 3: Design Options for Visualization of Time-Domain Data [5]

Category	Criterion	Design Options	
Time	Temporal primitive	<i>time points</i>	<i>time intervals</i>
	Structure of time	<i>linear</i>	<i>branching</i> <i>cyclic</i>
Data	Frame of reference	<i>abstract</i>	<i>spatial</i>
	Number of variables	<i>univariate</i>	<i>multivariate</i>
	Level of abstraction	<i>data</i>	<i>data abstractions</i>
Representation	Time dependency	<i>static</i>	<i>dynamic</i>
	Dimensionality	<i>2D</i>	<i>3D</i>

This table will now be discussed, while explaining the choices made for the implementation of the innovative time-domain cells of the Time-Domain Filtered-Monte-Carlo (TD-FMC). The first category of design choice pertains to the modeling and representation of time. In the TD-FMC, the elementary points of the signals (similar to pixels for images) will designate time instants rather than time intervals. Time will be viewed as linear, rather than cyclical (as in the Spiral representation) or branching (branches stem from time instants, corresponding to different scenarios or designs).

The second category of design options for the visual mining of signals corresponds to how the data (the responses) is modeled. For the TD-FMC, the data pertaining to the design of dynamic systems is multivariate and in the more general cases, it does not depend on geospatial position, therefore falling in the “abstract” frame of reference type. As explained previously, clustering of the data is not desirable for the TD-FM, which makes the desire level of abstraction refer to data instead of data abstractions.

The last category of design options pertains to how the data is visually represented. One option consists in performing dynamic animation of the data, i.e. presenting a snapshot of the dataset for a time instant, and then advancing in time. The user can thus get a feel of the time evolution of the system. However, as pointed out in Aignier et al. [5], animation loses its value as the number of dimensions of the data increases. Therefore, for the implementation of the TD-FMC, a static representation of the time-domain data will be used, in which the full time behavior of the dataset is visualized at once. Finally, the data can be represented in three dimensions, as seen in Figure 35, or in two dimensions. As explained previously, three-dimensional visualization can help get a grasp of the global trends, but the fact that data can easily be hidden hinders further interaction with the data. Thus, a bi-dimensional representation of the data will be adopted in the TD-FMC, similar to the data representation in the original Filtered Monte-Carlo technique for static metrics.

2.1.4.2 Description of the Innovative Visualization Environment

The previous section described the possible design options and the first choices for the development of the visualization environment that will enable the implementation of the Time-Domain Filtered-Monte-Carlo. This environment, whose purpose is to help investigate and explore the design space for dynamic systems subject to dynamic transient constraints, will be named the Visual Transient Response Explorer (VisTRE). As explained after the statement of Hypothesis 1, VisTRE will contain the multivariate scatterplot matrix (showing the input design parameters and the static responses), to which will be added the new cells visualizing the time-domain signals.

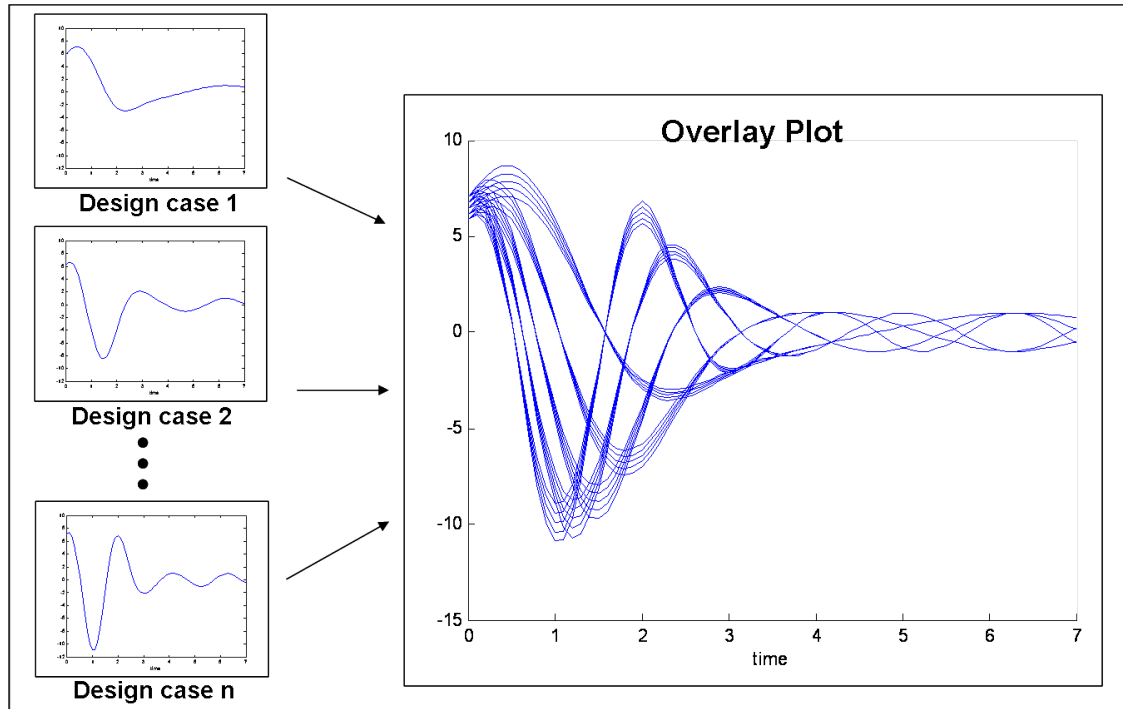


Figure 39: Overlay Plot for One Time-Domain Cell of VisTRE

Based on the design criteria derived above, overlay plots are chosen as the solution for the implementation of those new cells. In these, each design case is no longer represented by a point in the cell, but by a behavior curve. Each cell corresponds to a particular signal and a particular transient event, and plots the clustering of all the behavior curves that are taken by this signal when the design space is entirely sampled

through a Monte-Carlo simulation. A notional overlay plot for the visualization of the transient response of a dynamic signal is shown in Figure 39.

The environment resulting from the combination of the time-domain cells and the scatterplot matrix for static parameters is shown in Figure 40. In this figure, only one design scenario is represented, in order to illustrate one of the fundamental concepts of ViSTRE: a design case (or design scenario, or design point in the design space) is represented as a point in each cell of the scatterplot matrix, while it is visualized as a curve in the time-domain cell, which exhibit the transient behavior of the system.

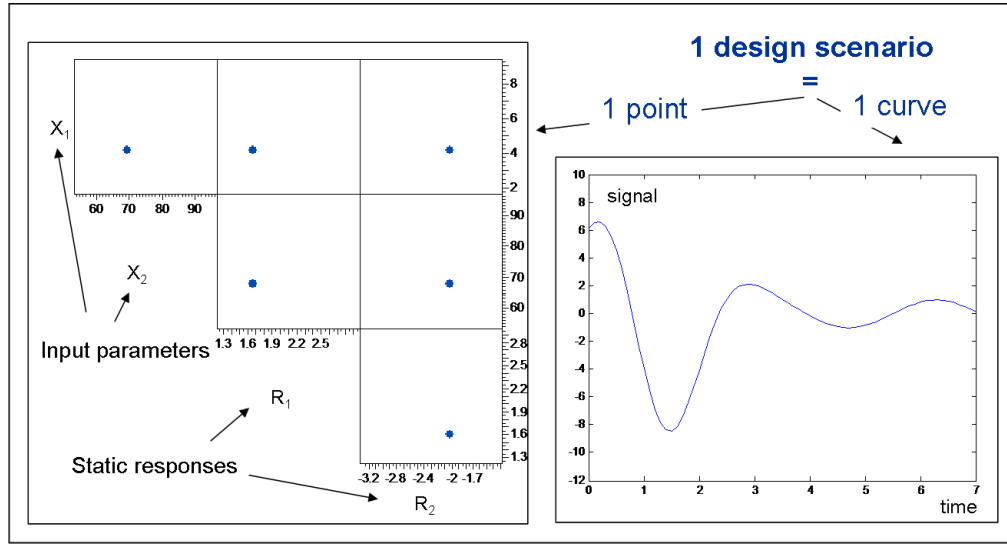


Figure 40: ViSTRE: Visualization of One Design Scenario

In order to thoroughly explore the design space, the first step of the Time-Domain Filtered-Monte-Carlo was to run a Monte-Carlo simulation that produces the static and dynamic responses for a vast number of design scenarios. Then, the data can be imported in ViSTRE so that the full design space can be visualized and explored. The resulting environment is illustrated in Figure 41.

The time-domain part and the static cells are interactively linked so that the selection of a behavior curve in the time-domain cell triggers the selection of the corresponding points in the static cells, and vice versa. In each of these time-domain cells, the user can then add or modify dynamic constraints. In the figure shown

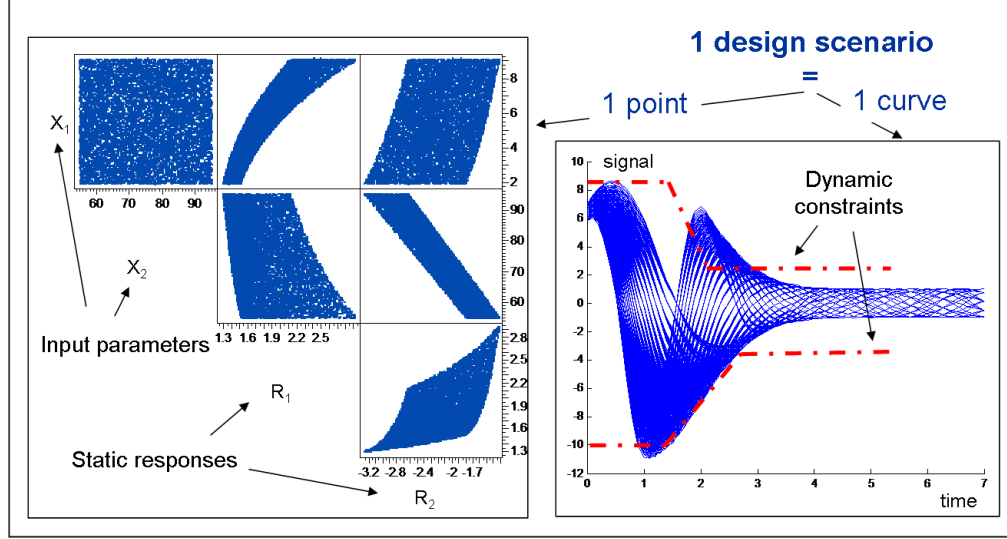


Figure 41: VisTRE: Visualization of the Full Design Space

above (Figure 41), dynamic constraints of the shape of the power quality constraints discussed in chapter 1 are shown as the red dashed lines.

The interactivity of the environment will then allow the user to visualize the behavior curves that meet (or do not meet) those constraints, as illustrated in Figure 42, where the design scenarios that violate the constraints are represented as red curves in the time-domain cells and as red points in the static scatterplot. Finally, as in the original Filtered-Monte-Carlo, the user can filter the design space so as to keep only those design scenarios that meet the dynamic constraints. This is illustrated in Figure 43, where a sizeable portion of the design space (first cell X_1 vs. X_2) has been removed, and where all the dynamic signals on the right hand-side remain within the required bounds.

In addition to the new capabilities for filtering dynamic constraints, the original capability of the Filtered-Monte-Carlo to filter design scenarios based on static constraints is still an option as well. Thus, when used in conjunction with the static multivariate scatterplot, the time-domain scatterplots enable to select and visualize the design scenarios that meet the static constraints as well as the dynamic ones.

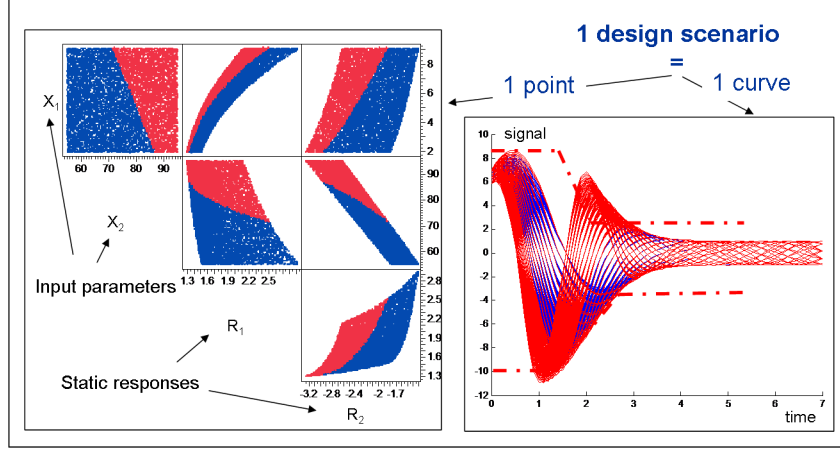


Figure 42: VisTRE: Visualization of Undesirable Design Scenarios

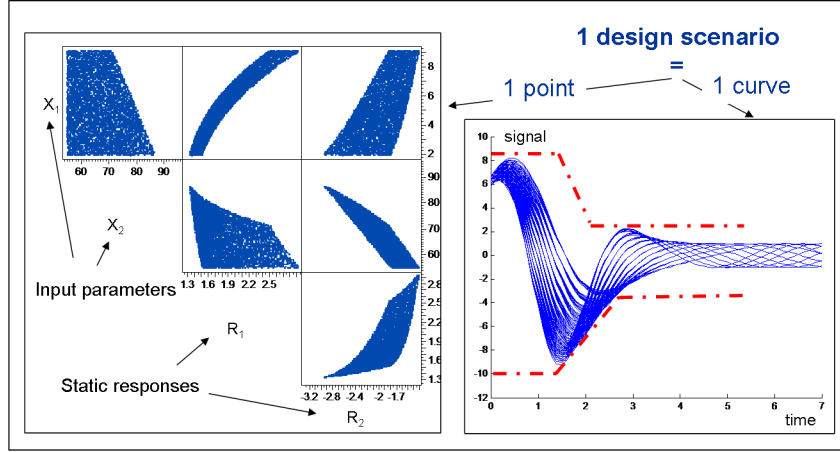


Figure 43: VisTRE: Visualization of the Design Space after Filtering

Further optimization can be carried on from there, using the static scatterplot matrix in the same way as in the original Filtered-Monte-Carlo.

In the first chapter, it was explained that there may be uncertainty on the values of the constraints, whether they be static or dynamic. In VisTRE, the addition or modification of a constraint is practically done instantly. Thus, when VisTRE is used as the visualization end of the Time-Domain Filtered-Monte-Carlo, the user can rapidly investigate the effect of changing constraints on the design space.

In a nutshell, after the Monte-Carlo Simulation, the resulting data is imported into the innovative interactive visualization environment VisTRE. Then, a few simple steps are performed in order to obtain a set of design solutions that meet the static constraints as well as the dynamic ones. The procedure for the design space exploration in VisTRE of dynamic systems subject to dynamic transient constraints is summarized in Figure 44.

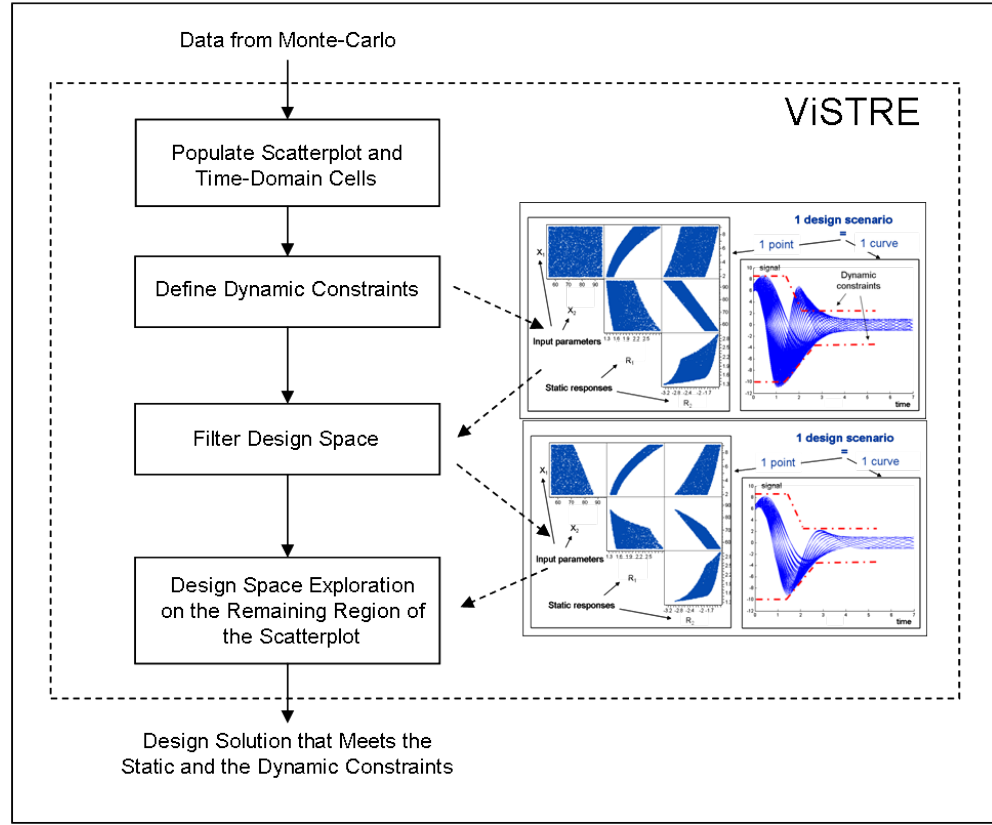


Figure 44: Using VisTRE for Design Space Exploration of Dynamic Systems: Algorithm

2.1.4.3 Generation of Time-Domain Signals: a Challenge for the Implementation of the Time-Domain Filtered-Monte-Carlo

At the conceptual and preliminary design stages, the time-domain behavior of the dynamic system is usually obtained through TDS, as explained in the previous chapter. Hence, the behavior curves generated by the Monte-Carlo Simulation for the

proposed Time-Domain Filtered-Monte-Carlo technique should ideally be computed using the TDS models.

However, as explained in Chapter 1, the use of TDS for design space exploration is limited by the slow run times that TDS models often exhibit. Even when the transient regime of interest only spans a very short time interval, TDS still needs to compute the behavior of the dynamic system leading to the steady-state point at which the transient is triggered. This is illustrated in Figure 45.

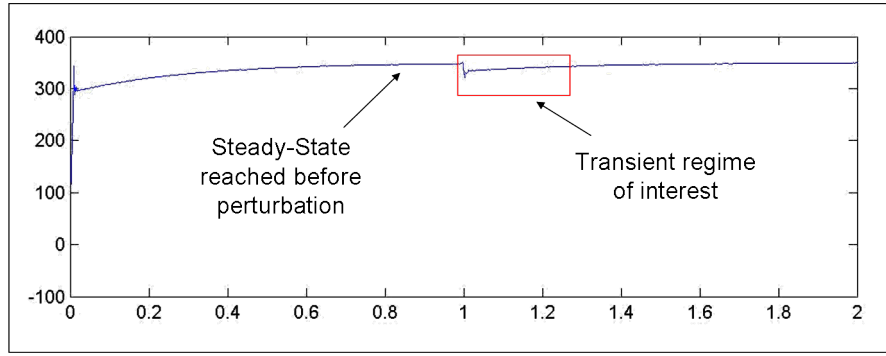


Figure 45: Time-Domain Simulation: Settling Time Before Perturbation

Because the Filtered-Monte-Carlo technique typically requires thousands of runs in order to thoroughly sample the design and operation space, these limitations in the simulation time of traditional TDS models pose a hindrance to the application of the proposed Time-Domain Filtered-Monte-Carlo technique. Therefore, in order to generate the points and curves in the Time-Domain Filtered-Monte-Carlo, the simulation process needs to be made more efficient. This leads to the formulation of the following research question, the answer to which will be a major area of focus for this thesis:

Research Question 2 (RQ2): How can one make the system simulation process more efficient in order to speed up the optimization/verification of dynamic systems and facilitate the implementation of the proposed Time-Domain Filtered Monte-Carlo approach?

2.2 *Advanced Modeling and Simulation of Dynamic Systems*

In the first part of the present chapter, an innovative data farming approach, the Time-Domain Filtered-Monte-Carlo (TD-FMC), was formulated for the thorough exploration of the design space of dynamic systems. In the TD-FMC, the design space is thoroughly sampled via Monte-Carlo Simulation, which produces a vast dataset, corresponding to the values of static responses and to the behavior curves of dynamic responses. Finally, because of limitations of TDS, advanced modeling of dynamic systems is needed in order to make the Monte-Carlo Simulation more efficient and manageable. In this part, options for efficient modeling and simulation of dynamic systems are investigated, after a brief overview is given on traditional modeling methods, in the context of the enforcement of dynamic constraints.

2.2.1 Modeling of Dynamic Systems

In order to simulate their behavior, dynamic systems need to be modeled mathematically. The type of mathematical model that is used for simulation greatly depends on the type of system and on the purpose of the simulation.

The most general form of the mathematical model of a dynamic system with a dynamic response y is given by Equation 3, where $\bar{\mathbf{x}}$ is the vector of design and operation variables (that may vary with time).

$$y = f(\bar{\mathbf{x}}, t) \tag{3}$$

For more clarity, in this section, $\bar{\mathbf{x}}$ will only designate the design characteristics of the system (such as resistance value, capacitance). The operation variables, which are the inputs of the system with fixed design characteristics, will be designated by the vector $\bar{\mathbf{u}}$. In this section, the term “input” will designate the operation variables, unless specified otherwise. These operation variables, such as the power consumption

demand from a motor, typically vary with time. The model of the dynamic system can then be rewritten:

$$y = f(\bar{\mathbf{x}}, \bar{\mathbf{u}}, t) \quad (4)$$

However, in most complex systems, the function f is unknown. The systems are modeled by differential equations, to which f is a solution. For example, a single input and single output system can generally be represented by the following n th-order differential equation:

$$F\left(y, \frac{dy}{dt}, \dots, \frac{d^n y}{dt^n}, u, \frac{du}{dt}, \dots, \frac{d^m u}{dt^m}, t\right) = 0 \quad (5)$$

2.2.1.1 Classification of Time-Domain Models

Depending on the modeling choice, the properties of the differential equation of Equation 5 vary. Kulakowski et al. gives a good overview of the different options available to the designer for the modeling and simulation of dynamic systems [89].

One important classification category relates to the linearity of the system. A system is linear when f (in Equation 4) is linear with respect to the parameter $\bar{\mathbf{u}}$, i.e. the principle of superposition applies:

$$f(\bar{\mathbf{x}}, \bar{\mathbf{u}}_1 + \bar{\mathbf{u}}_2, t) = f(\bar{\mathbf{x}}, \bar{\mathbf{u}}_1, t) + f(\bar{\mathbf{x}}, \bar{\mathbf{u}}_2, t)$$

In this case, the function F of the differential equation (Equation 5) is linear with respect to x and y . Multivariate Linear systems have the interesting property that they can be analyzed by considering them as a superposition of single variable systems:

$$f(\bar{\mathbf{x}}, u_1, u_2, \dots, x_l, t) = f(\bar{\mathbf{x}}, u_1, 0, \dots, 0, t) + f(\bar{\mathbf{x}}, 0, u_2, \dots, 0, t) + \dots + f(\bar{\mathbf{x}}, 0, 0, \dots, u_l, t)$$

An essential consequence of linearity is that for each operation variable, the response of the system can be entirely derived from the response to certain types of operation inputs. For instance, if one knows the response of the system to pure sine inputs for all frequencies, then one can derive the system response to any input signal that can be decomposed into a sum of sine waves via the Fourier transform. Or, if one is interested in the response of the system to any input step function, one only needs to know the response for one value of step function.

The second type of modeling option pertains to the spatial dependence of the model: if the system response depends on the spatial coordinates x, y, z , then the model is said to be distributed, and lumped otherwise. In the case of a “distributed” model, the system is represented by Partial Differential Equations (PDE). Otherwise, the system is called a “lumped system”, and Ordinary Differential Equations (ODE) are used. A widely used example of distributed models are the Navier-Stokes equations for the modeling of fluid motions in fluid mechanics.

Equation 5 is an example of a differential equation describing a system whose characteristics vary with time. Such a system is called “time-varying”. A system that is not time-varying is called “stationary” (or “time-invariant”). An example of stationary system is a circuit with constant resistance. If the resistance varies with time, then the circuit is a time-varying system. In the stationary case, the system is modeled by a differential equation with coefficients that are constant in time. In the example of the single input/single output system described by Equation 5, the differential equation for a stationary system becomes:

$$F\left(y, \frac{dy}{dt}, \dots, \frac{d^n y}{dt^n}, u, \frac{du}{dt}, \dots, \frac{d^m u}{dt^m}\right) = 0 \quad (6)$$

Finally, if the dynamic response that is monitored is not continuous, then the system is called a “discrete system”. For example, a traffic light can be modeled as a discrete system, since its light color may only alternate between three states (red,

amber, green). Discrete systems are modeled by time-difference equations instead of differential equations.

Several types of classification for dynamic systems have been presented. They are summarized in Table 4, which is adapted from Kulakowski et al [89].

Table 4: Classification of System Models [89]

Criterion	Type of model	Description
Linearity	nonlinear	Principle of superposition does not apply
	linear	Principle of superposition applies
Spatial dependency	distributed	Dynamic response functions of spatial coordinates
	lumped	Dynamic responses independent of spatial coordinates
Time dependency	time-varying	Model parameters vary in time
	stationary	Model parameters are constant in time
Continuity	continuous	Continuous range of independent variables
	discrete	Distinct values of independent variables

2.2.1.2 State-Space Representation

The models discussed so far are traditional input-output models, where the variable that varies is the dynamic signal under consideration (and the coefficients of the differential equations in the case of time-varying systems). A powerful (and dominant) tool for the simulation of dynamic systems is provided by the state-space representation of systems [89]. The *state* of a dynamic system is the minimum set of variables q_1, q_2, \dots, q_n that is sufficient to describe the system at any instant in time: if one knows the values of the input for $t > t_0$ and the state of the system at the instant $t = t_0$, then one can determine the behavior (the future states) of the system at all instants $t \geq t_0$. The state variables q_1, q_2, \dots, q_n are grouped into the state vector $\bar{\mathbf{q}} = (q_1, q_2, \dots, q_n)$. For example, if the system of interest is a rock that is thrown in the air and if the response of interest is the position of the rock, then the vector

composed of the position and the velocity of the rock can be considered as a state vector.

The state vector is thus a combination of the dynamic responses of interest and other parameters such as their derivatives. In state-space representation, each response of interest (output of the model) can be written as an algebraic function of the state and input variables, as stated in Equation 7 [89].

$$\begin{aligned}
y_1 &= g_1(q_1, q_2, \dots, q_n, u_1, u_2, \dots, u_l, t) \\
y_2 &= g_2(q_1, q_2, \dots, q_n, u_1, u_2, \dots, u_l, t) \\
&\vdots \\
y_n &= g_n(q_1, q_2, \dots, q_n, u_1, u_2, \dots, u_l, t)
\end{aligned} \tag{7}$$

For a lumped-parameter system, the essential property of the state-space representation is that the system can be represented by a set a first-order differential equations, listed in Equation 7. A compact form is given in Equation 9.

$$\begin{aligned}
\dot{q}_1 &= f_1(q_1, q_2, \dots, q_n, u_1, u_2, \dots, u_l, t) \\
\dot{q}_2 &= f_2(q_1, q_2, \dots, q_n, u_1, u_2, \dots, u_l, t) \\
&\vdots \\
\dot{q}_n &= f_n(q_1, q_2, \dots, q_n, u_1, u_2, \dots, u_l, t)
\end{aligned} \tag{8}$$

$$\dot{\bar{\mathbf{q}}} = f(\bar{\mathbf{q}}, \bar{\mathbf{u}}, t) \tag{9}$$

Because of the latter property, state-space models are typically much easier to simulate than the input-output models represented by complex high-order differential equations. Simulating the system behavior using state-space theory consists of computing the state vector, modeled by Equation 9, and converting the state vector

into the output vector of interest, given by Equation 7. This is illustrated in Figure 46 [89], for a system with l input variables, n state variables (or states), and p output variables (the dynamic responses).

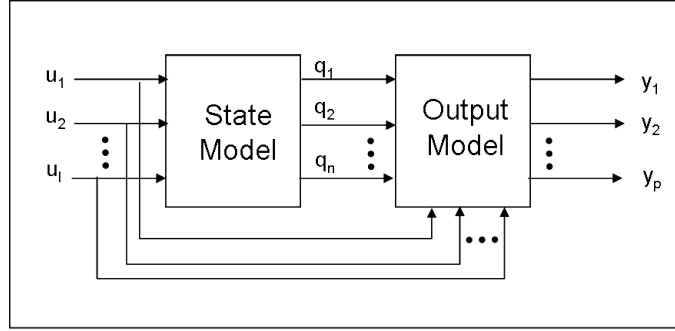


Figure 46: Block Diagram of a State-Space Simulation [89]

2.2.1.3 Frequency Domain Representation

Traditional input/output modeling with differential equations and state-space representations are two major ways of representing a dynamic systems. For Linear Time-Invariant systems (LTI), there is another way of converting the differential equations into a more convenient representation form: the frequency domain representation [46]. At the foundation of LTI theory lies the fact that the behavior of a system is entirely characterized by its impulse response, ie its theoretical response to the unit-impulse input [133]. The unit-impulse function $\delta(t)$ can be defined as the limit, when $\varepsilon \rightarrow 0^+$, of the functions $\delta_\varepsilon(t)$ defined by:

$$\delta_\varepsilon(t) = 0 \text{ when } t > \varepsilon, \text{ with } \int \delta_\varepsilon(t) dt = 1 \quad (10)$$

The impulse response of the LTI system is called the *transfer function* and is denoted $h(t)$, and the response of the system $y(t)$ to an input signal $u(t)$ is given by the convolution of $h(t)$ with $u(t)$ [133]:

$$\begin{aligned}
y(t) &= h(t) * u(t) \\
&= \int_{-\infty}^{\infty} h(\tau) \cdot u(t - \tau) d\tau
\end{aligned} \tag{11}$$

The frequency domain representation for LTI system is obtained by taking the Laplace transform of the transfer function. The Laplace transform at the frequency s is given by the Equation 12:

$$H(s) = \int_0^{\infty} h(t) \cdot e^{-st} dt \tag{12}$$

The advantage of using the frequency domain is that it is easier to interpret. In particular, many phenomena in dynamic systems (especially in control systems) contain combinations of frequencies [46]. The linearity of the system allows for the analysis of the effect of each frequency separately.

However, the frequency domain representation is limited to LTI systems. Even though many systems can in practice be considered as linear or be linearized around their points of operation, there are still many complex systems that cannot be treated as linear, especially in modern power systems that heavily rely on electronics. More importantly, dynamic constraints are defined in the time domain, and thus the behavior of the system is needed in the time-domain. If one chooses to study the frequency-domain behavior, one would need to then obtain the time-domain behavior by performing the inverse Laplace transform, which can be complex.

2.2.1.4 Focusing the Scope of this Research

As was seen in this section, there exist numerous types of dynamic systems and various ways of modeling them. It will be seen in Chapter III that transient responses often present nonlinear characteristics. The signals of interest are in general continuous, and time-varying. The vast majority of systems onboard an aircraft do not

present geospatial dependencies. Therefore, in this thesis, the dynamic systems under consideration will be modeled as nonlinear, time-varying, lumped and continuous.

As explained previously, in order to evaluate the compliance of the system with the transient dynamic systems, the time-domain response of the system needs to be computed. Thus, the frequency-domain representation will not be considered, and this thesis will focus on the modeling and simulation of dynamic systems in the time-domain, whether using input/output models with differential equations or state-space models.

Usually, dynamic systems are too complex to obtain an analytical expression of their behavior, and numerical methods need to be used. Numerically obtaining the time-domain behavior of the system from time-domain models is the essence of Time-Domain Simulation. The next section will briefly explore the most usual options for the implementation of TDS.

2.2.2 Methods for Time-Domain Simulation

There are two types of time-domain simulation: *continuous* simulation and *discrete* (or *discrete-event*) simulation [92]. They differ in how they take time into account. In continuous simulation, time is viewed as a continuous parameter, and models are generally represented by differential equations. The output of continuous simulation is therefore the full behavior of the system: the response of the system can be obtained at any instant t . Thus, computer continuous simulation generally involves the computing of an analytical solution to the differential equations. For most dynamic systems, the level of complexity prohibits the use of continuous simulation.

In discrete-event simulation, time is viewed as a discrete entity: the simulation is run at a finite number of time instants, and the state variables describing the system change instantaneously at those instants [92]. Therefore, a mechanism needs to be designed to decide the sequence of instants at which the states are computed.

There are two main categories of such time-advance mechanisms, defined by whether the time step between two events is fixed or varying. In the *fixed-increment* (or fixed-step) time advance, the time step between two events is predefined and constant over the simulation. The finer the time step, the more accurate the simulation results will be. However, if the system response does not vary much, the simulation will be run at many instants that will see little change in the state variables. These unnecessary runs may consume a lot of computing time. In *next-event* time advance mechanisms, the time step between two events varies. At every instant, a routine is called to compute the next time instant at which the model will be simulated [92].

Time-Domain Simulation typically marches in time to numerically solve differential equations, whether they are the high-order differential equations of the input/output models or (more commonly) the first order differential equations of the state-space representation. When the state-space representation is used, the numerical solving of differential equations is a process called integration. There are many integration methods found in TDS . Well-known examples of numerical integration methods are the Euler method and the Runge-Kutta method [89].

The Euler integration method was the first published numerical method for solving a first-order ODE of the following form:

$$\frac{dx}{dt} = f(x, t) \quad (13)$$

with the initial condition $x(t_0) = x_0$.

The Euler method is based on the following approximation of the derivative, with a time step Δt :

$$\frac{dx}{dt} \approx \frac{[x(t_0 + \Delta t) - x(t_0)]}{\Delta t}$$

This yields the numerical approximation of the solution:

$$x(t_0 + \Delta t) \approx x(t_0) + f(x_0, t_0) \cdot \Delta t \quad (14)$$

Figure 47 gives a geometrical interpretation of the approximation error introduced in the Euler method, induced by considering the derivative as constant between t_0 and $t_0 + \Delta t$.

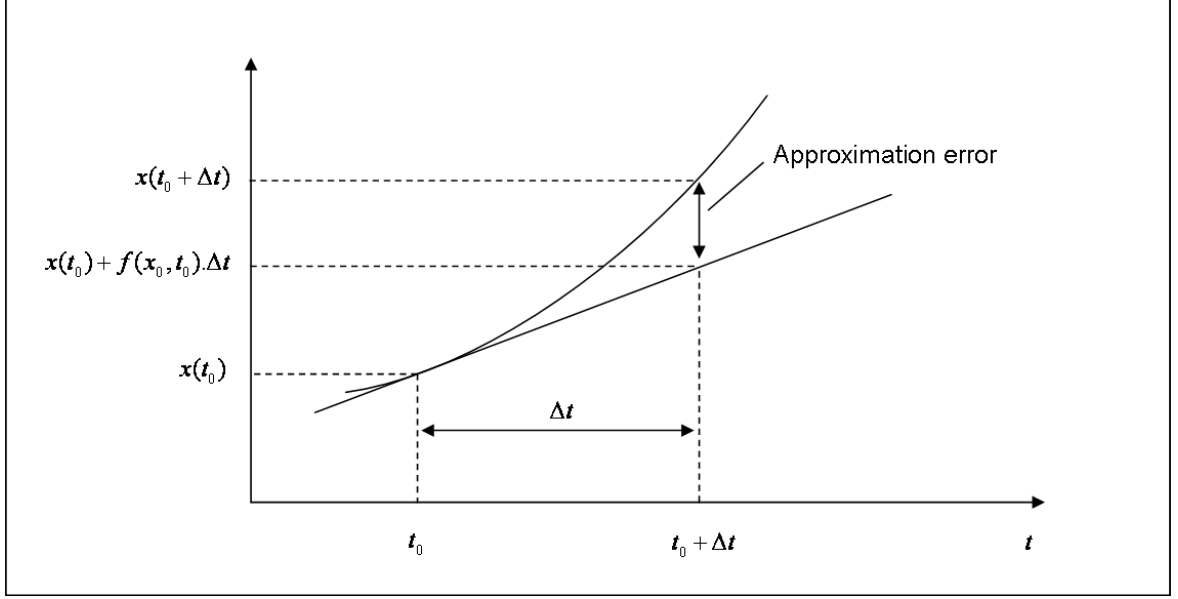


Figure 47: Approximation Error of the Euler Integration Method [89]

Modern numerical integration methods, such as the Runge-Kutta method, include higher-order terms of the Taylor series to approximate the derivatives. They have been shown to induce smaller approximation error, especially when the time step length decreases. In the Runge-Kutta methods, the solver explores intermediate points within the time steps in order to evaluate the higher-order derivatives [89]. The widely used fourth-order Runge-Kutta method uses four tentative steps between t_0 and $t_0 + \Delta t$, and the resulting approximation of the output parameter is given by:

$$x(t_0 + \Delta t) \approx x(t_0) + \left(\frac{\Delta t}{6}\right) \cdot (k_1 + 2k_2 + 2k_3 + k_4), \quad (15)$$

$$\begin{aligned}
\text{where } k1 &= f[x(t_0), t_0] \\
k2 &= f\left[x(t_0) + \Delta t \cdot \frac{k1}{2}, t_0 + \frac{\Delta t}{2}\right] \\
k3 &= f\left[x(t_0) + \Delta t \cdot \frac{k2}{2}, t_0 + \frac{\Delta t}{2}\right] \\
k4 &= f[x(t_0) + \Delta t \cdot k3, t_0 + \Delta t]
\end{aligned}$$

Implementations of numerical Time-Domain Simulation of dynamic systems are also characterized by other parameters, as seen in Kulakowski et al [89]. They include the initial and final simulation time, minimum and maximum step size, and error tolerance.

The set of implementation choices and options discussed above forms the core of a larger process whose goal is to simulate the behavior of a dynamic system through TDS. Law and Kelton formalized the process of performing a simulation study [92], as given in Figure 48.

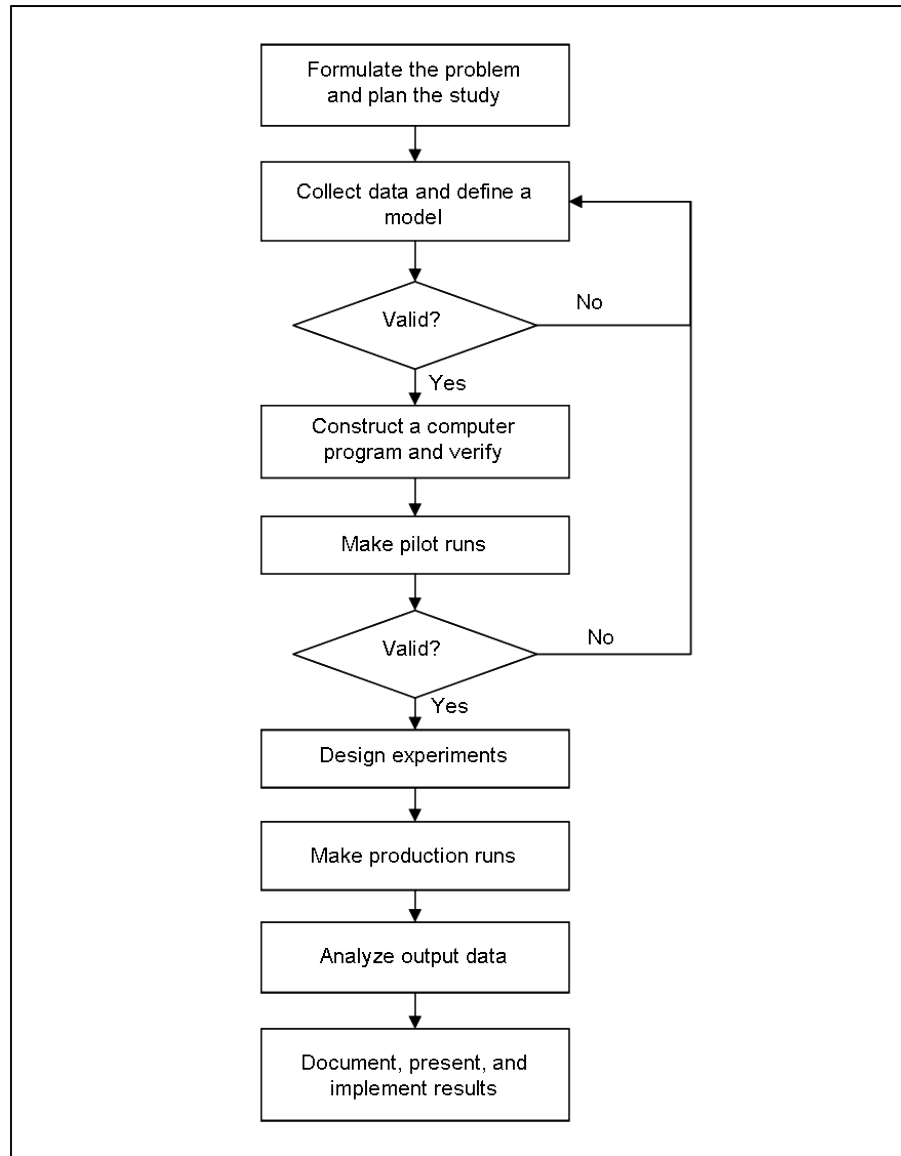


Figure 48: Steps in a Simulation Study [92]

This process outline is generic, and it should be noted that in real applications, alterations may be made to the process, or there may exist more iteration loops within the process. The process starts with the statement of the study objectives and a general planning. Then, data on the system of interest is collected, and the designer decides what modeling option is chosen to represent the system. First validation tests are performed on the model, and if successful, they lead to the next step, which is the development of a computer program for the simulation of the systems. This

simulation program is run on a few pilot cases, and if validation is successful, then the designer decides what design cases will be simulated. It is worth pointing out that this activity should have been partly done in the first step when planning the study. Indeed, a simulation environment is always developed with its possible future use in mind. After the experiments have been designed, the simulation program is actually run, performing what is termed “production runs”. This activity produces data that is analyzed in the next step, and finally, the results are presented or implemented in the next level of the overall system development process. Interestingly, when a vast number of production runs is performed, the last two steps correspond to the data mining and visualizing processes described in previous sections.

This section gave an overview of traditional methods for time-domain simulation of dynamic systems. As explained in Chapter I, TDS often is a time-consuming process, especially when the system under consideration is a complex system represented by higher-order differential equations. In order to produce the vast number of behavior curves required by the implementation of the proposed Time-Domain Filtered-Monte-Carlo methodology, the TDS process needs to be made more efficient.

2.2.3 Surrogate Modeling for Efficient Simulation

2.2.3.1 Surrogate Modeling of Static Responses

The challenge posed by time-domain simulation and other dynamic simulation tools was already existent for problems dealing with design and operation spaces composed of static parameters only. For example, in Phan et al. [127], the Filtered-Monte-Carlo was applied on an actual electrical test rig and each run would take dozens of minutes, therefore impeding the direct application of a Monte-Carlo simulation. To circumvent this obstacle, a *surrogate model* of the electrical test rig was created. A surrogate model of a tool, or meta model, is a parametric approximation of this tool, created by using statistical regression techniques on experimental data (or “training data”). Thus, a surrogate model consists of equations giving output metrics (the

“responses”) as a function of input parameters (the “variables”). These statistical approximation equations are generally simple to evaluate, which makes the surrogate models run quickly and efficiently. Therefore, the essential concept behind surrogate modeling is that the surrogate models are used as a substitute to the original Modeling and Simulation (M&S) tool, allowing the user to perform more runs at a cheaper cost (in terms of time, efficiency, cost). This is illustrated in Figure 49, where the original produces an output y that is a function of design and operation variables $(\bar{\mathbf{X}})$, and where a surrogate model produces an approximation of y , $\hat{y} = \hat{f}(\bar{\mathbf{X}})$. The approximation error is obviously given by $\varepsilon(\bar{\mathbf{X}}) = \hat{y} - y$.

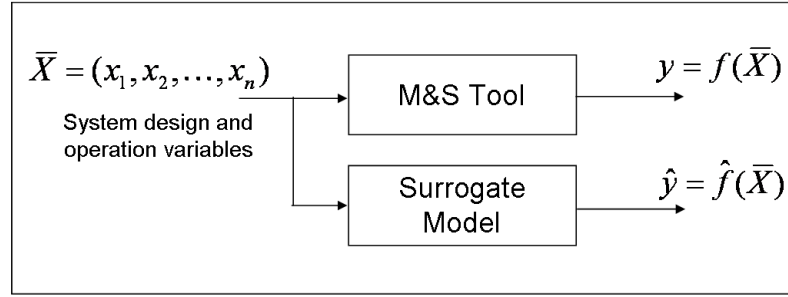


Figure 49: Static Surrogate Model

The generation of surrogate models is a multi-step process. A representative example is found in Kirby [83], where surrogate models of aircraft sizing and cash flow analysis tools are created in order to enable the thorough and efficient exploration of the design space for the aircraft. The first step is to define the problem: the responses of interest need to be identified, as well as the most influential design variables. Then ranges of variation for the design variables are defined. This is of special importance, because the surrogate model being an approximation of the M&S tool, it will only be valid within the ranges of variation for which it has been created, unless proved otherwise. The next step is to carefully select the runs that will be performed on the original M&S tool, and which will produce results that will serve to create the surrogate models. This activity is the subject of a formal theoretical domain, called

Design of Experiments (DoE), which will be detailed later in this thesis (Chapter IV). Then, these runs are performed in a “test campaign”, during which the M&S tool is run multiple times with different input settings (training cases). Then the mapping between the output and the inputs is created by performing a statistical regression, which minimizes the approximation error. There exist several regression techniques commonly used for surrogate modeling [33, 117]. These techniques can be categorized in two main branches: parametric regressions, which assume beforehand an equation form, and non-parametric regressions, which allow for more flexibility in the form of the regression equation. Finally, a validation test is performed in order to ensure that the surrogate model reproduces the behavior of the M&S tool in a satisfactory manner. This step is sometimes called “verification of the goodness of fit”, and in addition to verifying that the regression error is low for the training data, it generally includes performing additional runs of the original M&S tool for random settings of the design vector $\overline{\mathbf{X}}$, and verifying that the corresponding outputs closely match the outputs predicted by the surrogate models. The process for the generation of surrogate models is summarized in Figure 50.

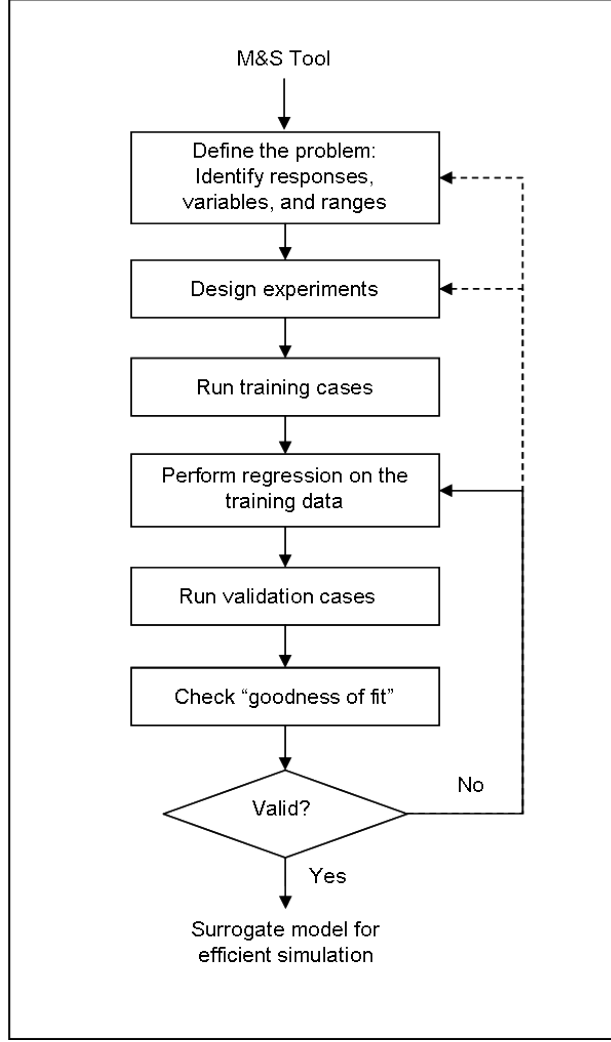


Figure 50: Generic Process for the Generation of a Surrogate Model

The regression on the training data produces simple equations that approximate the M&S tool. During this activity, the goal is to minimize the error between the responses predicted for the training cases by the surrogate model and the actual responses that were collected at the output of the original M&S tool. Typically, the metric quantifying the approximation error induced by the surrogate model is the Mean Squared Error (MSE) on the training dataset. The MSE, representing the average of the squared error for each training case, is defined as follows [152]:

$$\begin{aligned}
MSE &= E \left[\left(\widehat{\bar{\mathbf{Y}}} - \bar{\mathbf{Y}} \right)^2 \right] \\
&= \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2
\end{aligned} \tag{16}$$

where m is the number of training cases

$\bar{\mathbf{Y}} = (y_1, \dots, y_m)$ is the vector of responses calculated by m training cases

$\widehat{\bar{\mathbf{Y}}} = (\hat{y}_1, \dots, \hat{y}_m)$ is the corresponding vector of responses
predicted by the surrogate model

The two following examples of regression techniques, which are widely used for surrogate modeling, are instances of parametric regression and non-parametric regression.

Polynomial Response Surface Equations (RSE) are a class of parametric regression that is widespread for surrogate modeling [117, 118, 49, 33]. The principle is to assume that the response R follows a polynomial law. The most general form of a quadratic polynomial RSE is as follows :

$$R = b_0 + \sum_{i=1}^n b_i x_i + \sum_{i=1}^n b_{ii} x_i^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n b_{ij} x_i x_j \tag{17}$$

where n = number of design variables x_i

b_0 = intercept regression coefficient

b_i = regression coefficients for linear terms

b_{ii} = regression coefficients for pure quadratic terms

b_{ij} = regression coefficients for cross product terms

x_i, x_j = design variables or factors

Artificial Neural Networks (ANN) constitute a wide class of non-parametric regression techniques [66, 140, 33]. They have become a popular choice for the regression of nonlinear parameters, since it has been demonstrated that they can approximate any arbitrary continuous function [64, 140]. Moreover, they generally perform well when approximating discontinuities for integrable functions. ANN's are structured as a collection of interconnected simple processing units called *perceptrons*, *nodes* or *neurons*, which act as a transfer function, thus mimicking the way the brain functions. For each connection between two neurons, there is an associated weight representing the strength of the connection. A neuron's output will be the transform of the weighted sum of inputs, as shown in Figure 51 [32]. In addition to the weighted sum of inputs, an offset term (or intercept term) b is often added.

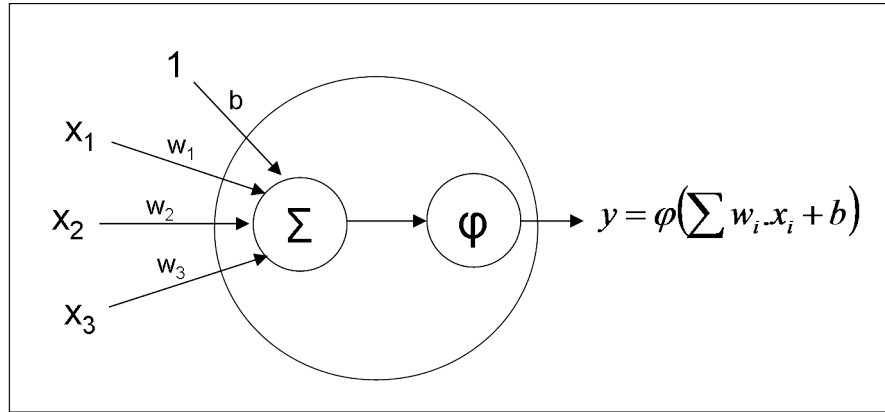


Figure 51: Schematics of a Neuron with Transfer Function ϕ

These neurons can be arranged in multiple ways to form more or less complex neural network architectures. Usually, within an architecture, neurons are organized into several groups called layers. When there is no feedback from a neuron layer to another, i.e. when all the neuron connections flow in the same direction, then the neural network is qualified as “*feedforward*”. In the presence of feedback connections, the neural network is termed “*recurrent*”. More details on recurrent neural networks are given in the next chapter.

In his seminal papers [64, 63], Hornik gave proof that a three-layer feedforward neural network architecture, with the middle layer comprised of neurons with sigmoid transfer functions (cf. Equation 18), can approximate any continuous (or integrable) function, provided that the number of neurons in the middle layer (called “hidden layer”) is sufficient and that the weights of the neuron connections are adjusted. The plot of the sigmoid function and a schematic of the feedforward ANN are given in Figures 52 and 53 respectively.

$$S(z) = \frac{1}{1 + e^{-z}} \quad (18)$$

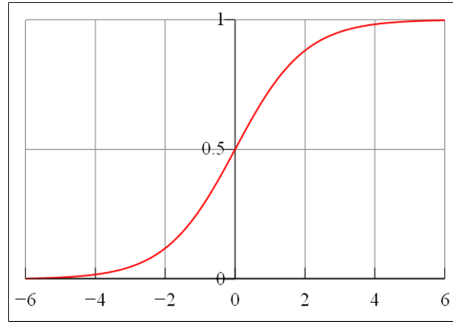


Figure 52: Sigmoid Function

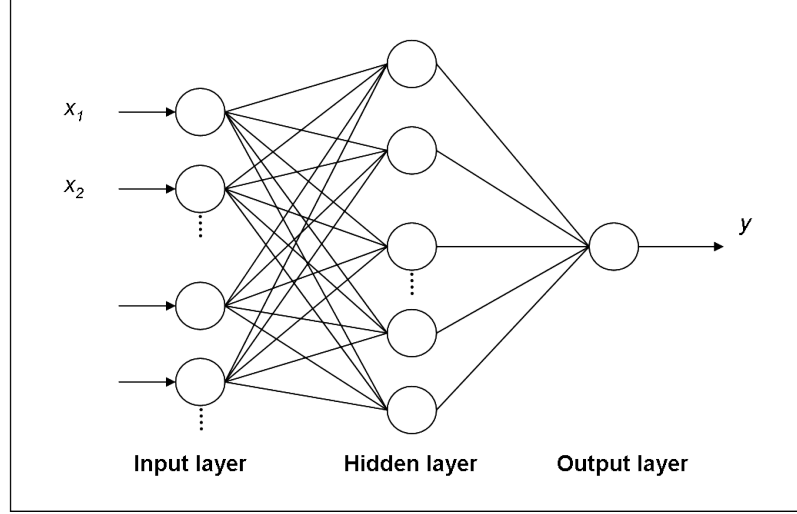


Figure 53: Example of Artificial Neural Network Architecture with One Hidden Layer

The output of a neural network is thus a response \hat{y} that depends on the values of the inputs and on the weights of the connections between neurons. For a three-layer feedforward network, the expression of the response \hat{y} can be given explicitly:

$$\hat{y} = d + \sum_{j=1}^{N_H} \left(c_{jk} \left(\frac{1}{1 + e^{-(b_j + \sum_{i=1}^N (a_{ij} \cdot x_i))}} \right) \right) \quad (19)$$

where N is the number of input design variables

x_i is the i^{th} design variable

a_{ij} is the weight of the connection from the i^{th} design variable to the j^{th} hidden node

b_j is the intercept term for the j^{th} hidden node

c_{jk} is the weight of the connection from the j^{th} hidden node to the output layer node

d is the intercept term for the output layer node

The training of a three-layer ANN consists in adjusting the coefficients a_{ij} , b_j , c_{jk} , d so that the total regression error for the training data (measured by metrics such as the MSE) is minimized. This training process, called *supervised learning* [85], is thus an optimization problem where the objective function is the total error, and the variables of the optimization are the network coefficients. There is a number of methods available to perform supervised learning, one of the most commonly used being the *back propagation* algorithm [130]. The essential principle of the latter resides in the computation, at each iteration, of the gradients of the regression error with respect to the weights for each neuron, and the use of these gradients to determine the new values of the weight for the subsequent iteration.

Because it generates statistical approximation equations that are quick to evaluate, surrogate modeling can be used to speed up simulation processes. The inherent efficiency of surrogate models is thus employed as an enabler for the efficient exploration of complex design spaces. In the case of static metrics, it has also been employed to simulate the thousands of design scenarios for the creation of the multivariate scatterplot in the Filtered-Monte-Carlo technique. This can be summed up in the following observation:

Observation 6: In the case of static metrics, the efficiency of static surrogate modeling can be leveraged in order to enable the efficient design/operation space exploration through the Filtered-Monte-Carlo approach.

2.2.3.2 *Dynamic Surrogate Modeling: Surrogate Modeling for Dynamic Responses*

It was shown in the previous section that static surrogate modeling may be used advantageously to speed up simulation processes dealing with static output responses. Because of the efficiency of these types of stochastic approximation models, they have proved to be an enabler for the implementation of the Filtered-Monte-Carlo technique for static responses, as shown in Ender [49], for the design of a system-of-systems, or

in Phan et al. [127], for the exploration of the operation space of an electrical test rig.

For design problems based on time domain simulation, the concept of dynamic surrogate modeling has been developed in order to speed up simulation times: here, the output of the surrogate models are dynamic responses instead of static ones. For example, in Balchanos [10], dynamic surrogate models were generated for elementary power system sub-components and then assembled in order to facilitate the simulation of the higher-level system.

The enabling capability offered by static surrogate modeling for the implementation of the Filtered-Monte-Carlo, coupled with the efficiency promises shown by dynamic surrogate models in terms of improvement of simulation times for TDS tools, allows for the formulation of the following hypothesis:

Hypothesis 2 (H2): Dynamic surrogate modeling of time domain simulation models will enable the efficient implementation of the Time-Domain Filtered Monte-Carlo technique for the exploration of design/operation spaces for dynamic systems subject to dynamic constraints.

The concept of dynamic surrogate modeling is illustrated in Figure 54. Evidently, the output response \hat{y} of the surrogate model is now an approximation function that depends not only on the inputs \overline{X} , but also on time t . It is worth noting that in the most general case, the input components may themselves be dynamic parameters, i.e. they depend on time. For instance, the aircraft electrical network voltage depends on operation variables such as the power consumption requirements of the electrical loads [127]. These power consumption requirements typically vary with time during the aircraft mission [55].

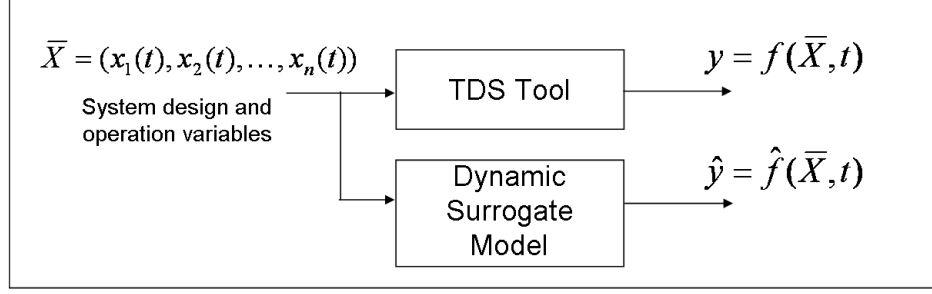


Figure 54: Dynamic Surrogate Model

As with static responses, the generation of dynamic surrogate models thus consists in determining a mathematical model of the TDS tool, based on a set of observed data signals for different design/operation scenarios. This activity, which deals with dynamic signals, forms the essence of a discipline called “system identification” [100].

System identification is a wide field that has been mainly used by controls theory, in the design of control systems for dynamic systems. For instance, in aircraft system design, system identification is a major activity in the design of control systems for flight control surfaces [148]. There are multiple sub-disciplines and approaches pertaining to system identification. Hence the following research question:

Research Question 3 (RQ3): What system identification approach is suitable for the generation of dynamic surrogate models in the context of transient time-domain simulation of dynamic systems subject to dynamic constraints?

This research question is investigated in the next chapter, which provides a literature review of the potential alternatives for a system identification approach that is suitable for the problem at hand.

2.3 *Summary: First Elements of the Methodology*

In this chapter, it was seen that optimization-based methods pose a difficulty to the thorough exploration of the design space for dynamic systems subject to dynamic transient constraints. Furthermore, these methods fail to provide the necessary flexibility to account for the potential uncertainty in those constraints. Therefore, a new methodology is needed for the integration of dynamic transient constraints in the design of dynamic systems. The quest for this new methodology necessitates the investigation of the following research question:

Research Question 1 (RQ1): How can one efficiently and thoroughly explore the design and operation spaces while accounting for dynamic responses and pertaining uncertain dynamic constraints?

The foundations of an innovative methodology have been formulated. This methodology builds on a new data farming approach formulated herein as an extension of the Filtered-Monte-Carlo technique to the time domain. The resulting Time-Domain Filtered-Monte-Carlo (TD-FMC) approach is the product of the first hypothesis of this thesis:

Hypothesis 1 (H1): A data farming approach, based on the integration of time as a dimension in the Filtered-Monte-Carlo approach, will conduce to the efficient and thorough exploration of the design/operation space for dynamic signals with dynamic constraints.

The TD-FMC uses a Monte-Carlo Simulation, which samples the entire design and operation space and produces a vast set of static design responses and time-domain signals, representing the transient behavior of the system across the design and operation space. These Monte-Carlo outputs are then imported into an innovative interactive visualization environment, which allows the designer to explore and query

the design and operation space. In this proposed Visual Transient Response Explorer (VisTRE), which handles a combination of the multivariate scatterplots for static parameters and overlay plots for transient time-domain responses, the designer can instantly define and modify dynamic constraints, and filter the design space so that only the design solutions that verify the dynamic transient constraints are kept for subsequent design refinement.

The Monte-Carlo Simulation activity of the TD-FMC produces a vast number of time-domain transient responses. Because of the limitations of Time-Domain Simulation explained in Chapter I, a new approach for the TDS of dynamic system needs to be formulated in order to enable the implementation of the TD-FMC. This is the issue raised in the following research question:

Research Question 2 (RQ2): How can one make the system simulation process more efficient in order to speed up the optimization/verification of dynamic systems and facilitate the implementation of the proposed Time-Domain Filtered-Monte-Carlo approach?

In existing implementations of the original Filtered-Monte-Carlo technique, static surrogate models, which are statistical approximations of Modeling and Simulation (M&S) tools, have exhibited the desired efficiency, thus enabling the efficient exploration of the design space. Therefore, it is hypothesized that dynamic surrogate models, which emulate the behavior of time-domain simulation tools, will enable the implementation of the TD-FMC:

Hypothesis 2 (H2): Dynamic surrogate modeling of time domain simulation models will enable the efficient implementation of the Time-Domain Filtered Monte-Carlo technique for the exploration of design/operation spaces for dynamic systems subject to dynamic constraints.

The activity of generating dynamic surrogate models is akin to performing system identification on the time-domain simulation model. Since there exist many approaches for the system identification of a dynamic system, it is necessary to investigate the third research question:

Research Question 3 (RQ3): What system identification approach is suitable for the generation of dynamic surrogate models in the context of transient time-domain simulation of dynamic systems subject to dynamic constraints?

The investigation of the latter research question will be done in the following chapter. The core elements of the methodology formulated in this thesis are summarized in Figure 55.

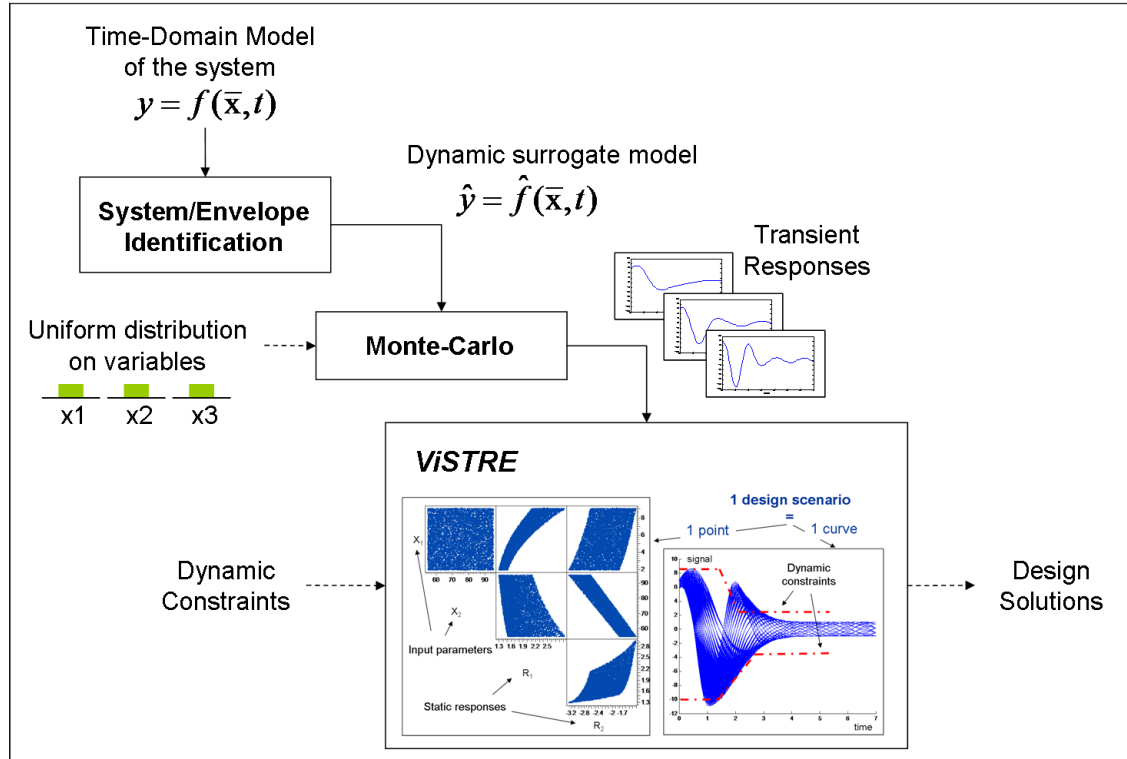


Figure 55: Time-Domain Filtered-Monte-Carlo: First Overview

Chapter III

SYSTEM IDENTIFICATION: CLASSIFICATION AND BENCHMARKING

As was seen in the previous chapter, one enabler of the interactive Time-Domain Filtered-Monte-Carlo will be dynamic surrogate models that approximate the time-behavior of the output signals of the dynamic systems; this activity relates to system identification. In order to determine potentially suitable system identification approaches for the generation of dynamic surrogate models, it is first necessary to determine the category to which the problem at hand belongs.

3.1 Characterization of the Problem

As was explained in the opening chapter of this thesis, the objective of this research work is the integration of transient-related constraints in the early design of aircraft dynamic systems. In this context, it is assumed that there exists a parametric time-domain simulation model of the system under consideration. In order to enable the efficient and thorough exploration of the design/operation space, one specific aim of this work is to formulate a methodology for the generation of surrogate models based on the TDS model of the dynamic system.

3.1.1 Number of Parameters

The first characterization criterion is the number of parameters involved in the models. This is further refined into the classification according to the number of input parameters (the variables) and the number of output parameters (the responses).

For aircraft dynamic systems, the design/operation space is generally highly multidimensional: there are multiple variables and multiple responses. Essentially, the problem thus belongs to the category of “Multiple Inputs Multiple Outputs” (MIMO) system identification problems [148]. However, it is assumed that the responses depend only on the inputs and on time and are mutually independent. Therefore, the problem can be decoupled into many subproblems: a multiple response model can be viewed as an aggregation of single response models [148]. Therefore, the system identification problem will be considered as a “Multiple Inputs Single Output” one (MISO).

3.1.2 Response Linearity

The nature of the response plays a fundamental role in the selection of the appropriate type of system identification. It was seen detail in the previous chapter that when the response behaves linearly with respect to its input parameters, the system is said to be *linear* and can be modeled by a set of linear differential equations, which have the advantage of being easier to solve than their nonlinear counterpart. It was also explained that the response of a linear system can be fully derived from its response to a few specific inputs. Even though rigorous linearity is practically impossible to achieve in physical systems, many dynamic systems may be approximated as linear systems around their typical operating points. Linear system identification is indeed a vast field of study that has generated many theories and applications [148]. However, in the present case, linearity cannot be assumed since transient regimes often exhibit nonlinear responses [60]. This is especially true in the design of today’s aircraft electric networks, which increasingly include highly nonlinear power electronics systems [115, 78]. The responses of the dynamic surrogate models to be created are therefore considered nonlinear.

3.1.3 Black Box vs. Grey Box Modeling

When a system identification model of a dynamic system is constructed, the type of approach depends of the level of prior knowledge that the designer has about the system. If the latter has sufficient knowledge to assume a mathematical form for the approximation model, the approach is termed “*grey-box*” approach [139]. In most cases, the designer cannot assume a form for the mathematical approximation of the dynamic system, and the system identification approach is qualified as a “*black-box*” one [139]. In system identification, it is generally considered as a rule to “*avoid estimating what is already known*” [139]. Therefore, a grey-box approach should be used when possible.

Because of the need for flexibility for the generation of dynamic surrogate models of dynamic systems subject to transient dynamic constraints, the system identification approach will be a “black-box” one. However, black-box approaches that can be readily adapted into grey-box ones will be favored.

3.1.4 Static Nature of Input Parameters

It was stated earlier that the inputs of a dynamic system may be dynamic. The example of the voltage response of an aircraft electrical network to the time-varying power consumption of the various loads was given. Another example is the attitude response of a flight control surface to commands from the pilot.

However, in the course of the design of these dynamic systems, designers typically limit their field of study to critical design/operation cases that will constitute the extremes of the design/operation space. For example, in the case of the electrical network behavior, only the voltage response to the step increase or ramp increase of load power consumption is characterized [52, 127]. An example of a generic test carried out on the electrical test rig modeled in Phan et al. is given in Figure 56 [127]. The test shown here aimed at analyzing the impact of levels of load consumption

and associated transient perturbations on the 350 VDC network voltage. The loads in presence were three different Electrohydrostatic Actuators (EHA), an Air Cycle Machine (ACM), a Recirculation Fan (RF), and a Programmable Load (PL).

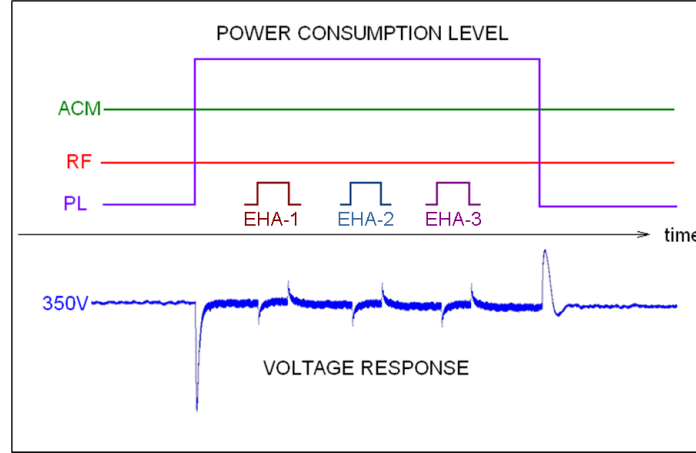


Figure 56: Transient Perturbations for the Analysis of an Electrical Test Rig [127]

In most cases, if not all, extreme design/operation scenarios can be fully characterized with static metrics. For example, a step increase is characterized by the time at which the step occurs, and by the final level that is achieved. Similarly, a ramp increase is characterized by the time at which the ramp occurs, the slope of the ramp, and the final level that is achieved or the time at which the ramp stops. This is illustrated in Figure 57.

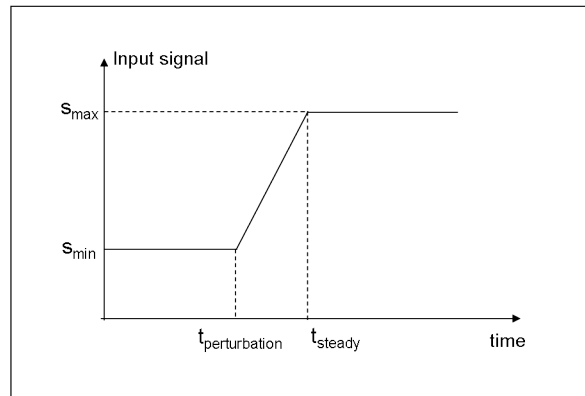


Figure 57: Characterization of a Dynamic Ramp with Static Parameters

Therefore, it will be assumed in this thesis that dynamic surrogate models with static input variables will be sufficient to enable the efficient and thorough exploration of the design/operation space of dynamic systems via the Time-Domain Filtered-Monte-Carlo technique. This can be restated as follows:

Assumption: In the context of an efficient and thorough design/operation space exploration of a dynamic system subject to transient dynamic constraints, critical design/operation scenarios can be characterized by static input variables.

Consequently, the input variables of the dynamic surrogate models that are to be generated will be static parameters. This is illustrated in Figure 58.

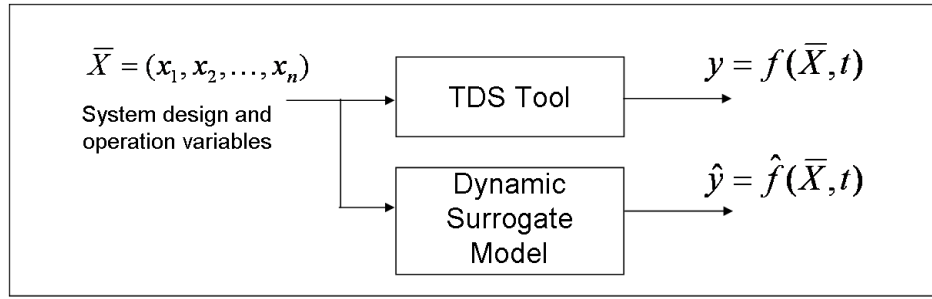


Figure 58: Dynamic surrogate model with static variables

3.1.5 Multiscale Nature of the Responses

Finally, it is important to look more closely at the nature of the responses. These have been identified as nonlinear above. Besides their nonlinearity, another major characteristic is the multiscale aspect of signals. The dynamic response signal may indeed exhibit certain behavior at short time scales (high frequency) while it will show another trend at longer time scales (low frequency). This is illustrated in Figure 59, where a signal is decomposed into a signal with low frequencies and a signal with high frequencies. Thus, when considering large time scales, the signal appears smooth and with relatively little oscillation. On the other hand, if one considers small time scales, which has the effect of “zooming in”, the signal will appear as highly oscillatory.

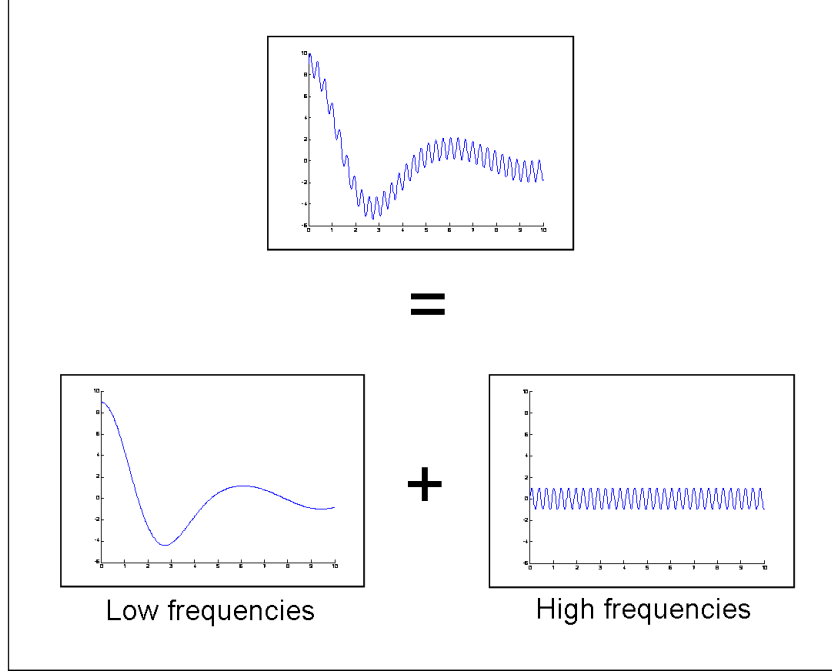


Figure 59: Decomposition of a Multiscale Signal

An example found in industrial applications is described in Felix and Routex [52], where the steady-state voltage of a 350VDC network exhibited high-frequency components induced by the high-frequency switching of power electronics involved in the control systems. This is illustrated in Figure 60, where at high time scales, the signal might appear constant (350 VDC), but at low time scales it will appear highly oscillatory around 350VDC and with spikes (corresponding to the transient undervoltage induced by load consumption perturbations).

The multiscale aspect of dynamic signals is, therefore, a major area of investigation of research endeavors such as the Vivace European project [4]. As emphasized in de Weck et al.[37], it poses a challenge to the simulation of dynamic systems, as multiscale models require small time steps as well as long simulation time windows in order to capture the various time scales.

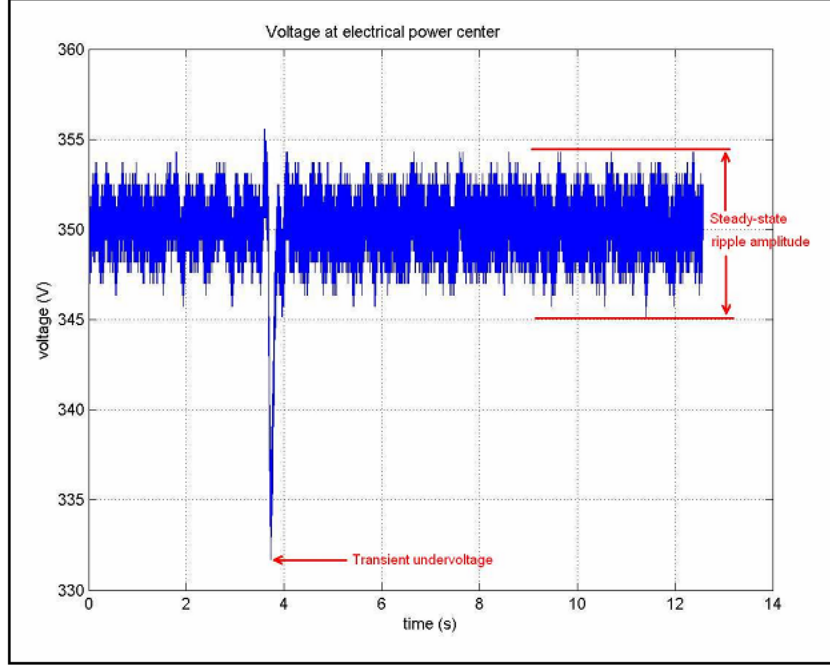


Figure 60: Steady-state Voltage of a 350 VDC Network [52]

3.1.6 Simulation Time and Transient Perturbation Time

An important parameter describing a Time-Domain Simulation run is the simulation time, which is the simulation clock time at which the simulation will stop. When simulating a dynamic system, the designer typically makes a few pilot runs to determine the order of magnitude for the length of the transient regime (in the milliseconds, in the seconds, in the minutes, etc.). Then the simulation time for the rest of the study is set accordingly.

In this thesis, it is assumed that the designer knows the order of magnitude for the length of the transient regime. Moreover, *the simulation time t_f will be fixed* across all the design space, in such a way as to ensure that the simulation encompasses the entire transient regime throughout the design space. This is not unreasonable, since the designer may make pilot runs beforehand, and since the dynamic transient constraints typically impose a duration limit on the length of the transient regime. Therefore, if for a simulated case the simulation time is large enough to encompass the

transient constraint but too short to encompass the totality of the transient response, the designer can reject this design case.

Finally, when simulating a dynamic system for the purpose of analyzing its transient response, the designer generally forces a transient perturbation to occur during the simulation. The time t_{pert} at which this transient event is triggered is controlled by the designer, as will be the case in this thesis.

3.1.7 Summary and Desired Characteristics

As a summary, the generation of dynamic surrogate models for the exploration of the design/operation space of dynamic systems subject to transient dynamic constraints necessitates a nonlinear black-box system identification approach. The models to be generated are MISO, with static input parameters. Finally, the selected system identification approach should enable the dynamic surrogate models to capture the potentially multiscale properties of the dynamic responses.

In addition to the above mentioned features, the selected system identification approach should ideally meet certain desirable criteria. Firstly, the training time should remain manageable. It is usually a given that the generation of surrogate models may be a long and difficult process, which is acceptable if the resulting efficiency of the models yields high payoffs in term of design/operation space exploration capabilities. However, the speed of the training process should be kept to reasonable levels.

3.2 *Benchmarking*

This section presents a review of the literature on the most promising system identification approaches for the problem at hand: these approaches are described and assessed. The comparative assessment, or benchmarking, is performed according to several criteria, derived from the discussion above.

These criteria are listed below:

1. *Universal nonlinear approximator*: this measures the ability of the system identification approach to model any nonlinear response with acceptable accuracy.
2. *Multiscale capturing*: this quantifies the ability of the system identification approach to capture the multiscale properties of the dynamic signal.
3. *Training speed*: this is a metric for the time it takes for the training process to converge.
4. *Algorithm complexity*: this is a metric quantifying the structural complexity of the training algorithm.
5. *Adaptability to grey-box formulation*: this relates to the capability of the black-box system identification approach to use prior knowledge about the behavior of the dynamic system.
6. *Surrogate simulation speed*: this metric quantifies the time it takes for the resulting system identification model to run the dynamic surrogate model. Since the purpose of surrogate modeling is to accelerate the TDS process in order to facilitate the Monte-Carlo Simulation, this criteria is of utmost importance.

3.2.1 Feedforward Neural Networks

Artificial Feedforward Neural Networks (FFNN) were introduced and extensively described in the previous chapter. It was seen, from the proof given by Hornik [63], that when structured with three layers and with sigmoid transfer functions in the hidden layer nodes (or neurons), they can approximate any continuous nonlinear function (and more generally, any integrable function).

Using a FFNN as a system identification approach for the generation of a dynamic surrogate model means treating the time t as an input parameter of the FFNN, as illustrated in Figure 61.

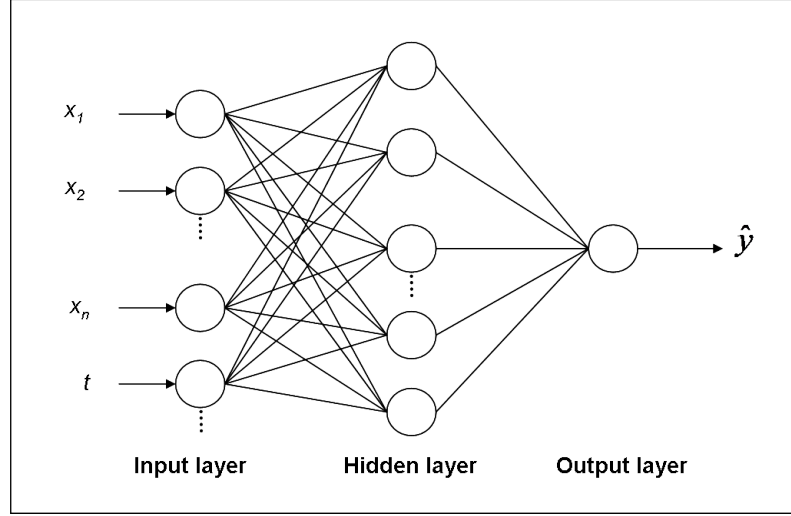


Figure 61: Time-Domain Feedforward Neural Network

Training a FFNN can be done using the Backpropagation algorithm, which is an optimization algorithm that is quite efficient. Because at each iteration of the algorithm, the error is computed for all the time samples for all the training cases, the training speed greatly depends on the number of time samples resulting from the multiple training runs of the original TDS model. The training speed also depends on the complexity of the architecture, i.e. on the number of nodes in the hidden layer.

This number of nodes, which is tuned by trial-and-error or by more sophisticated iterations, drastically increases for high frequency signals. In order to show the latter statement, the approximation with FFNN of a simple time series, with no other input parameter for the network, will be considered here. For example, a simple sine wave function over three periods, shown in Figure 62, is identified. The FFNN architecture approximating this time series is shown in Figure 63. One can see that the only input node is the one corresponding to the unique input t .

Because the transfer function of the hidden nodes are sigmoid functions (the “S curve”), the approximating output \hat{y} is a linear combination of sigmoid functions. Since one sigmoid function only has one inflexion point, this means that a signal with n inflexion points will require at least n hidden nodes to be approximated correctly.

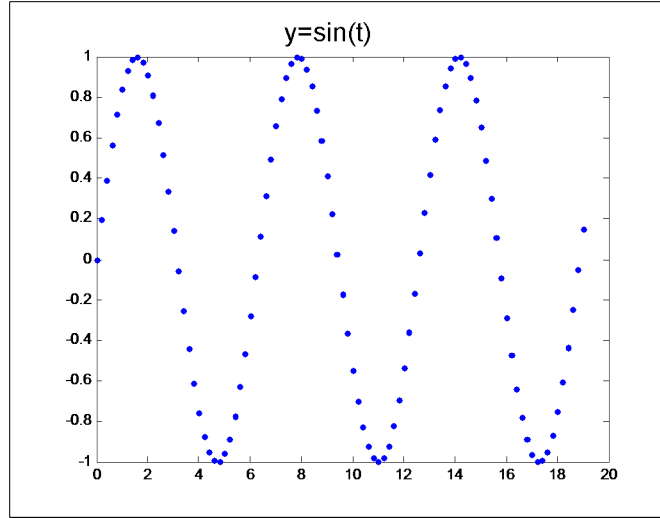


Figure 62: Sine Wave

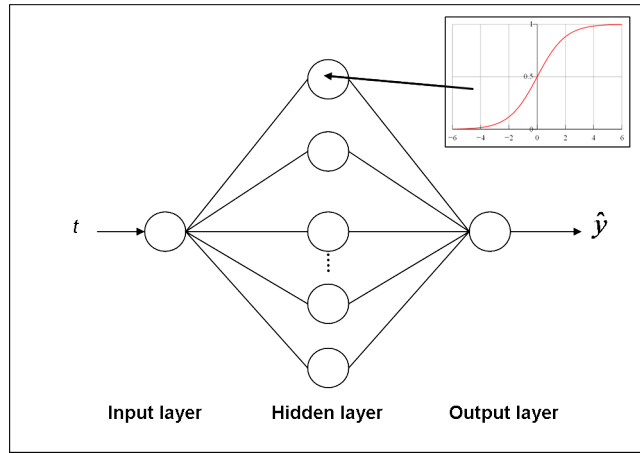


Figure 63: Feedforward Neural Network for Approximating a Simple Time Series







Or, one can see it as each maximum peak requiring at least two sigmoid functions (one ascending S curve, and one descending S curve) to be approximated. For the sine wave example, this means a minimum of 7 nodes for only 3 periods. For multiscale signals with high frequency components such as the controlled voltage shown in Figure 60, the required number of nodes for satisfactory system identification increases drastically, even more so when the other inputs are added to the neural network.






The FFNN formulation is adaptable to grey-box modeling by tweaking the transfer function of the neural network nodes. For example, if one knows that the output is

a product of some term with the sine of t , then the transfer function of the output node of the neural network can be multiplied by $\sin(t)$.

Dynamic surrogate models generated by FFNN are equations that are simple functions of time and the other input variables. They are thus very efficient to run.

Table 5: Feedforward Neural Network Assessment

Criteria	Feedforward Neural Network
1. Universal Nonlinear Approximator	
2. Multiscale capturing	
3. Training speed	
4. Algorithm complexity	
5. Adaptability to grey-box formulation	
6. Surrogate simulation speed	

Excellent
Very Good
Good
Fair
Poor

3.2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) form a popular class of system identification approaches for nonlinear responses. Based on their ability to model dynamic signals and store time-dependent effects, they have been applied to a wide diversity of problems related to dynamic systems, as seen in Medsker and Jain [108], or in Luo and Unbehauen [101].

In the field of system identification for control of dynamic systems, Narendra and Parthasarathy showed in their seminal publication ([121]) that the use of RNN's was a viable option. Lee et al. showed that RNN's perform better than feedforward ANN's in approximating and identifying nonlinear signals [94]. Simple examples of application of RNN's for the identification and control of dynamic systems are given in Sjöberg [138].

Other examples of applications of RNN's include signal filtering and spectral estimation for the detection of seismic events [147], language learning in the domain of speech recognition software [14], electric load forecasting [28] and financial prediction

[56]. Interestingly, in Liang et al. [135], RNN's were successfully applied to synthesize the sounds of traditional Chinese music instruments.

Because they present feedback loops within their neural architectures, the output state of neurons may be dependent on the previous states of the system. This gives the RNN the ability to store memory of previous states, and thereby to capture long term effects in dynamic behaviors [108, 18]. Any neural network with a feedback loop is a RNN, which implies that the number of possible types of RNN architectures is virtually limitless [111]. In the Hopfield Network, the most general RNN architecture, all the neurons are connected to each other. Figure 64 shows a Hopfield Network with four neurons [130].

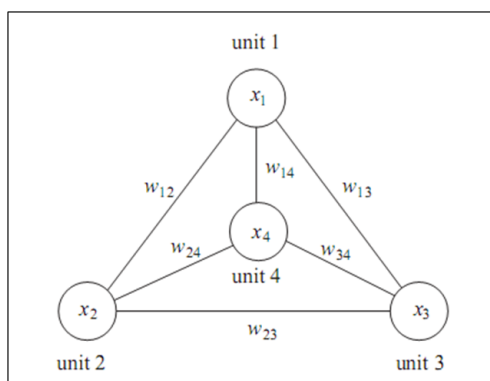


Figure 64: Hopfield Network with Four Neurons [130]

As in the feedforward case, RNN's may be organized as multilayer networks. The most general complex multilayer architectures, called Fully Recurrent Neural Networks see all their neurons connected to each other, and are thus a particular case of Hopfield networks. Figure 65 shows a more simple yet very popular type of RNN called Output Feedback Recurrent Neural Network [108].

One can see on Figure 65 that there is one feedback loop from the output layer to the input layer. This feedback loop connects the output to two “delay neurons” in the input layer, which store the values of y at the two previous time iterations: the

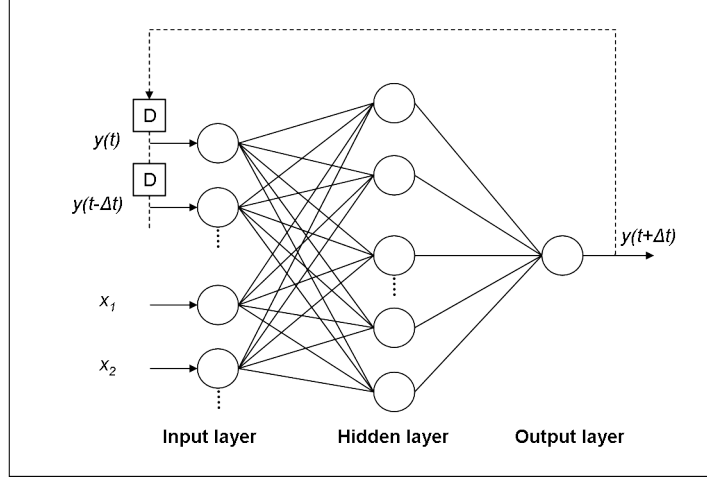


Figure 65: Example of Recurrent Neural Network Architecture

output y at time $t + \Delta t$ thus depends on its past values at instants t and $t - \Delta t$. Additionally, the dynamic output y is a function of the static inputs x_1 and x_2 .

It has been shown that many types of RNN architectures are capable of exhibiting good nonlinear system identification properties. In particular, Jin et al. [72] proved that a particular type of RNN architecture, called Globally Recurrent Neural Network architectures, can be considered as a universal approximator for nonlinear dynamic signals.

The learning process of a RNN can become challenging. Most of the well-established learning algorithms derive from gradient-descent techniques such as the backpropagation one discussed in a previous section. The backpropagation learning technique is fairly straightforward to implement. However, it is an optimization approach that has proved prone to lead to local minima, depending on the initial values given by the optimizer [108]. As summarized by Bengio et al. in the title of their article [15]:

“Learning long-term dependencies with gradient descent is difficult”.







Bengio et al. concludes that training RNN using gradient descent algorithms (such as the backpropagation technique) becomes increasingly difficult as the length of the simulation time window increases. Therefore, the bulk of past research efforts on RNN






learning techniques has focused on extending and improving the backpropagation technique for RNN, as reviewed by Pearlmutter in his comprehensive survey [126]. The work exhibits RNN gradient-descent learning algorithms that show acceptable properties in terms of learning speed and computational complexity, while still not achieving the performance of backpropagation with feedforward ANN's.

Besides gradient-descent learning algorithms, other optimization techniques have been developed to train RNN's. Some have shown good results in terms of efficiency of the training process, such as the second-order methods presented in Dos Santos and Von Zuben (third chapter in Medsker and Jain [108]). These optimization algorithms are based on improved versions of the conjugate gradient method.

From the discussion above, it is clear that RNN's have levels of performance that can differ according to the chosen type of architecture and learning method. However, average trends can be extracted so as to assess them with respect to the benchmarking criteria derived above. The resulting assessment is summarized in Table 6.

Table 6: Recurrent Neural Network Assessment

Criteria	Recurrent Neural Networks
1. Universal Nonlinear Approximator	
2. Multiscale capturing	
3. Training speed	
4. Algorithm complexity	
5. Adaptability to grey-box formulation	
6. Surrogate simulation speed	

Excellent
Very Good
Good
Fair
Poor

3.2.3 Meijer's Dynamic Neural Networks

In the previous section, a review of recurrent neural networks was performed. It was explained that training RNN's may be challenging; although novel methods have improved the efficiency of learning for RNN's, convergence rates and computational complexity are still inferior to those for feedforward neural networks. For this reason,

any nonlinear system identification approach built on feedforward ANN's is preferable at a first glance.

Meijer, in his PhD thesis [109], has proposed a new formulation of feedforward ANN's that is capable of approximating nonlinear dynamic signals. Meijer's Dynamic Neural Networks (MDNN) were developed in the context of electric and electronic circuit modeling. After observing that the large simulation times of TDS models of complex circuits impeded the thorough and efficient behavioral analysis and optimization of these circuits, Meijer proposed his MDNN's, which allowed him to replace TDS circuit models by approximation models. Because these approximation models treat the circuit holistically instead of as an aggregation of individual components, this process is referred to as "macro-circuit" modeling in the field of electric and electronic circuit analysis [43]. In essence, this is equivalent to generating surrogate models of the dynamic system represented by the circuit.

MDNN's are extensions of multilayer feedforward artificial neural networks. Their particularity lies in the fact that the neurons represent differential equations, thus enabling the capture of the circuit's dynamics. The differential equation governing the output y_{ik} of the i^{th} neuron of the k^{th} layer is given in Equation 20 below:

$$\tau_2(\sigma_{1,ik}, \sigma_{2,ik}) \frac{d^2 y_{ik}}{dt^2} + \tau_1(\sigma_{1,ik}, \sigma_{2,ik}) \frac{dy_{ik}}{dt} + y_{ik} = \mathcal{F}(s_{ik}, \delta_{ik}) \quad (20)$$

where:

the σ 's and the δ 's are characteristic parameters of the neuron

the τ 's are "timing functions" set prior to the learning process

\mathcal{F} is a nonlinear transfer function, typically a sigmoid

s_{ik} is the weighted sum of the outputs of the preceding layers connected to neuron ik .

For the i^{th} neuron of the k^{th} layer, the weighted sum s_{ik} is given by Equations 21 and 22:

$$s_{ik} \triangleq \mathbf{w}_{ik} \cdot \mathbf{y}_{k-1} - \theta_{ik} + \mathbf{v}_{ik} \cdot \frac{d\mathbf{y}_{k-1}}{dt} \quad (21)$$

$$= \sum_{j=1}^{N_{k-1}} w_{ijk} y_{j,k-1} - \theta_{ik} + \sum_{j=1}^{N_{k-1}} v_{ijk} \frac{dy_{j,k-1}}{dt} \quad (22)$$

where:

w_{ijk} and v_{ijk} are connection weights between the neuron j of layer $k-1$ and the neuron i of layer k

θ_{ik} is an offset parameter.

The MDNN's therefore represent a cascading set of differential equations and can be graphically described as shown in Figure 66.

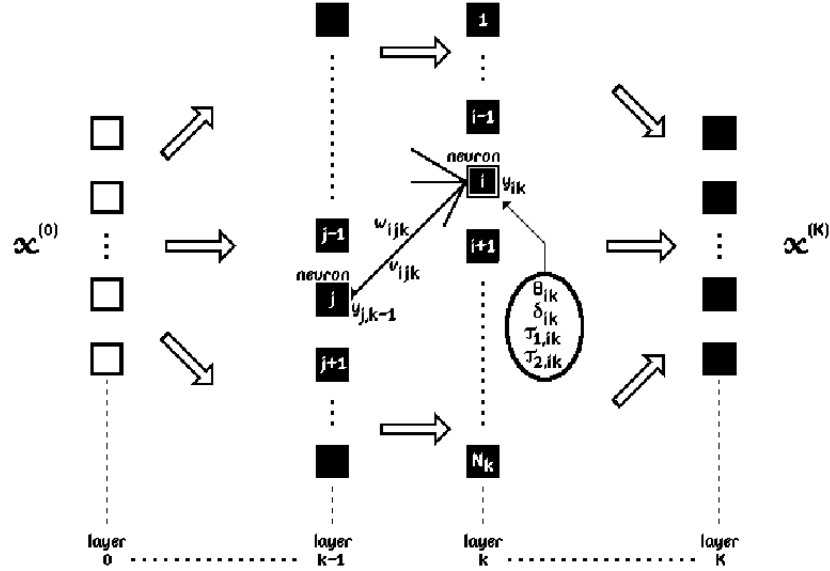


Figure 66: Schematics of Meijer's Dynamic Neural Networks [109]

During learning, the connection weights w_{ijk} and v_{ijk} , along with the neuron characteristic parameters θ_{ik} , δ_{ik} , $\sigma_{1,ik}$ and $\sigma_{2,ik}$, are adjusted using a supervised learning

approach, so that the input-output mapping approximates the training data. This amounts to a larger number of parameters per neuron than for the common feedforward ANN's, but since MDNN's are feedforward ANN's as well, the learning process should yield better converging rates than those of the RNN's discussed in the previous section. Meijer suggests using an extension of the traditional backpropagation technique, associated to a gradient-descent optimizer such as the conjugate-gradient method.

However, since the neurons represent differential equations, these need to be solved at each iteration of the learning process. The time is thus discretized and a time integration numerical solver such as the backward Euler scheme has to be employed [9]. This considerably penalizes the learning time and the computational complexity. The fact that the user can choose the discretization time step and that neurons can be dedicated to different kinds of dynamics (long-term vs. short term) makes the MDNN's fairly good solutions for multiscale identification.

Meijer showed that his MDNN's can be considered as universal approximators for linear dynamic signals, as long as they are quasi-static in nature, i.e. their dynamic is so fast that at any instant, the dynamic signal will have the same value across the entire circuit. For example, under the quasi-static approximation, the current has the same intensity across the circuit, regardless of whether it is an alternating or a direct current. In practice, this quasi-static condition is almost always verified since the characteristic length of aircraft circuits is typically far smaller than the wavelengths of signals [142]. In the case of nonlinear dynamic signals, Meijer explains that feedback loops from the output neuron to the input neurons may need to be added, thus transforming the MDNN's into dynamic RNN's with differential equations as neurons. In this case, learning can be heavily penalized since the drawbacks of RNN training reappear.






Extensions of the MDNN's were proposed by [23, 128] to make the formulation more general. This was done by proposing alternative nonlinear functions for the transfer function \mathcal{F} of Equation 20. It is also worth noting that allowing the transfer function \mathcal{F} to be customized for each neuron enables the learning process to use a priori knowledge about the circuit/system. This is a step towards grey-box modeling if needed.

The main drawback of the MDNN approach is that it produces a set of differential equations, albeit of lower orders, approximating the original TDS models. Therefore, in order to run a dynamic surrogate model generated with MDNN's, one needs to solve these differential equations, using either analytical tools or numerical integration methods such as those described in Chapter II. The surrogate run times can therefore be high, which might severely prevent the MDNN's from being a good enabler for the Monte-Carlo Simulation.

The results of the assessment of Meijer's Dynamic Neural Networks are summarized in the Table 7.

Table 7: Meijer's Dynamic Neural Network Assessment

Criteria	Meijer's Dynamic Neural Networks
1. Universal Nonlinear Approximator	
2. Multiscale capturing	
3. Training speed	
4. Algorithm complexity	
5. Adaptability to grey-box formulation	
6. Surrogate simulation speed	

Excellent
Very Good
Good
Fair
Poor

3.2.4 Wavelet Neural Networks

In the 1980s, a new mathematical tool was developed for signal processing: the wavelet transform theory, for which a thorough introduction may be found in Daubechies

[35]. Wavelets were originally formulated to circumvent the shortcomings of spectral estimation tools, such as the Fourier transform, in terms of localization in time of frequency components. More on the topic will be given further. Their elegance and power quickly ensured their popularity. Wavelets have since been used in a wide array of applications, related to the decomposition, filtering, compression, and reconstruction of dynamic signals. Perhaps the most well-known of such applications is the creation of the JPEG2000 image compression standard [144].

The system identification community has fostered the capabilities of the wavelet transform and combined them with neural network techniques to create a new and powerful nonlinear system identification approach: the Wavelet Neural Network (WNN), also called “Wavelet Network” or more simply “Wavenet” [77]. Before explaining the principles of wavenets, a brief overview of the underlying wavelet transform theory is provided.

3.2.4.1 The Wavelet Transform

It is well-known that the Fourier transform decomposes a signal into a basis generated by the sine and cosine functions. The wavelet transform acts in a similar fashion, but with a different basis, generated by a single function, which belongs to a specific family: the wavelets.

The term “wavelet” originates from its French translation “ondelette”, which means “little wave”. As this indicates, a wavelet is a function that satisfies two main conditions [145]:

1. its energy is localized in a short time window
2. it contains some oscillations in time

There are numerous types of wavelets. Figure 67 shows the plots of two popular examples of wavelets: the Morlet wavelet and the Mexican Hat wavelet.

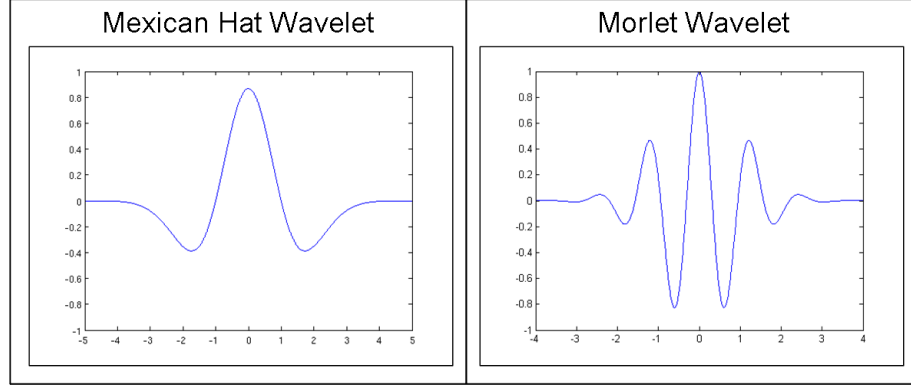


Figure 67: Mexican Hat Wavelet and Morlet Wavelet

The Wavelet transform theory has established that any $\mathcal{L}^2(\mathbb{R})$ function can be decomposed into a basis formed by any wavelet ψ , called the “mother wavelet”, and derived wavelets obtained by translation and dilation of the mother wavelet. These derived wavelets $\psi_{a,b}$, called “daughter wavelets”, are thus each characterized by a dilation (or scaling) parameter a and a translation parameter b , as expressed in Equation 23.

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) \quad (23)$$

The concepts of dilation and translation are illustrated in Figure 68, where a mother wavelet is plotted along with daughter wavelets resulting from a dilation, a translation, and a combination of both.

From the wavelet transform theory, a fundamental result is that any signal f can be approximated by a truncated portion of the series of the wavelet decomposition. This is expressed in Equation 24:

$$f(t) \approx \sum w_{a,b} \psi_{a,b}(t) \quad (24)$$

Reducing the original function f to its set of wavelet coefficients $w_{a,b}$ is a process called “Wavelet Transform” (WT), or “Wavelet Decomposition”. The reconstruction of the original signal from a series of wavelet coefficients and wavelet functions is

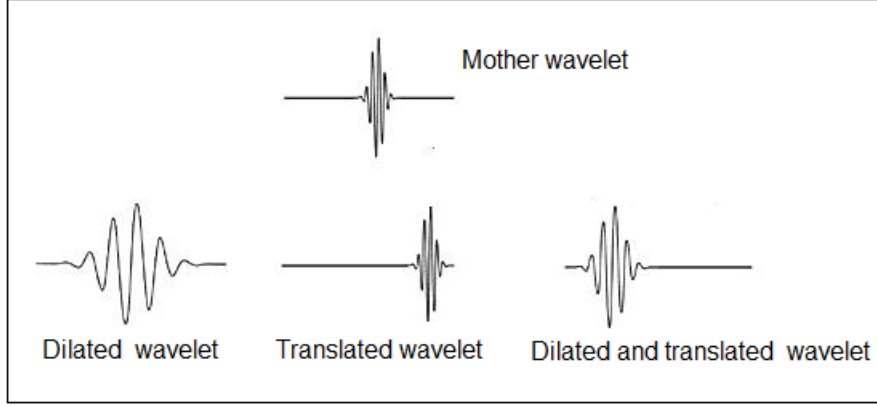


Figure 68: Dilation and Translation of the Mother Wavelet

called the “Inverse Wavelet Transform” (IWT). This terminology is similar to that of the Fourier theory, where Fourier Transforms compute series of Fourier coefficients, and where Inverse Fourier Transforms reconstruct the original function from the set of Fourier coefficients.

In the wavelet decomposition expressed in Equation 24, the dilation and translation parameters vary in a continuous fashion. The wavelet decomposition is thus called the “Continuous Wavelet Transform” (CWT). In the same way that the Discrete Fourier Transform discretizes the sine functions basis according through a discrete set of frequencies, the dilation and translation parameters can be discretized to form a discrete lattice of possible values. The wavelet transform process is then called “Discrete Wavelet Transform” (DWT), and the associated signal reconstruction process is called “Inverse Discrete Wavelet Transform” (IDWT). A formal introduction on wavelet theory is given in Appendix A.

When one looks at Figure 68, one can start to apprehend that the dilation factor of a wavelet relates to the frequencies captured by that wavelet. Hence the alternate term “scaling”. Similarly, the translation factor of a wavelet characterizes the location in time of the frequency component. Therefore, the wavelet decomposition of a signal can be viewed as the localization in time of the various frequency components

constitutive of the signal [35]. This is the area where the wavelet transform is clearly advantageous compared to the Fourier transform, since the latter can only give the frequency components present within a time window.

To help the understanding of what time-frequency localization means, one can make the analogy with music sheets. Scaling is equivalent to varying the pitch of a note, while translation relates to the instant at which the note is played. Performing a wavelet decomposition is thus analogous to transcribing a piece of music into a music sheet.

The ability to perform time-frequency localization is highly relevant for the analysis of transient signals, since the onset of a transient event will typically generate frequency components that quickly vanish. Thus, wavelet analysis has become a prime tool for transient and fault detection in power systems, as seen in Misrikhanov [113], in Lee et al. [93], and in Hua and Fang [67].

Time-frequency localization is an enabler for the multiscale analysis of dynamic signals. Indeed, the terms in the wavelet decomposition of Equation 24 can be re-ordered so that increasing the number of terms included in the truncated approximation series is equivalent to considering smaller dilation parameter values. In other words, refining the time scales is done by including more terms in the wavelet decomposition approximation. Thus, the wavelet decomposition of the signal can be viewed as the aggregation of secondary signals, each with a different grain of time resolution[24]. This process is called Multiresolution Analysis (MRA) and was formalized in Mallat [103].

So far, only the one-dimensional case has been treated. The results have been extended to multidimensional cases with signals in $\mathcal{L}^2(\mathbb{R}^n)$ (one of the dimensions being time). For instance, Kugarajah and Zhang have constructed multidimensional wavelets that are frames for $\mathcal{L}^2(\mathbb{R}^n)$ [88]. The construction of multidimensional wavelets is straightforward; there are a few alternative options, the most simple of

which is the tensor wavelet, obtained by multiplying the one-dimensional wavelets. The resulting multidimensional wavelet is given by Equation 25:

$$\bar{\psi}_{\mathbf{a},\mathbf{b}}(\bar{\mathbf{x}}, t) = \psi_{a_0,b_0}(t) \cdot \prod_{i=1}^N \psi_{a_i,b_i}(x_i) \quad (25)$$

where N is the number of design variables x_i

a_0 is the dilation parameter for t

b_0 is the translation parameter for t

a_i is the dilation parameter for x_i

b_i is the translation parameter for x_i

$\mathbf{a} = (a_0, a_1, \dots, a_N)$ and $\mathbf{b} = (b_0, b_1, \dots, b_N)$

An example of multidimensional wavelet is illustrated in Figure 69. The wavelet shown here was obtained by the tensor product of the Gabor wavelet [31].

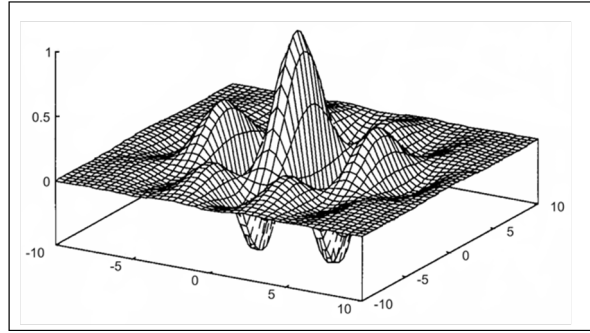


Figure 69: 3D Gabor Wavelet [31]

Thus, for the multivariate case, the dynamic signal can now be approximated by Equation 39.

$$f(\bar{\mathbf{x}}, t) \approx \sum_{k=1}^{N_w} w_{\mathbf{a}_k, \mathbf{b}_k} \bar{\psi}_{\mathbf{a}_k, \mathbf{b}_k}(\bar{\mathbf{x}}, t) \quad (26)$$

where N_w is the number of wavelets in the approximation

3.2.4.2 Wavelet Neural Networks for System Identification

The popularity of the wavelet transform theory prompted its extension to the field of signal approximation and system identification [77]. The wavelet decomposition of a signal f formulated in Equation 39 is readily transposable into a feedforward artificial neural network formulation, in which the neurons, now called wavelons, correspond to the daughter wavelets $\bar{\psi}_{a,b}$, as formulated by Zhang and Benveniste [162]. A schematic of a multidimensional wavelon, with a mother wavelet ψ , is given in Figure 70. A wavelon is thus defined by the mother wavelet and by the scaling and dilation parameters a_i and b_i .

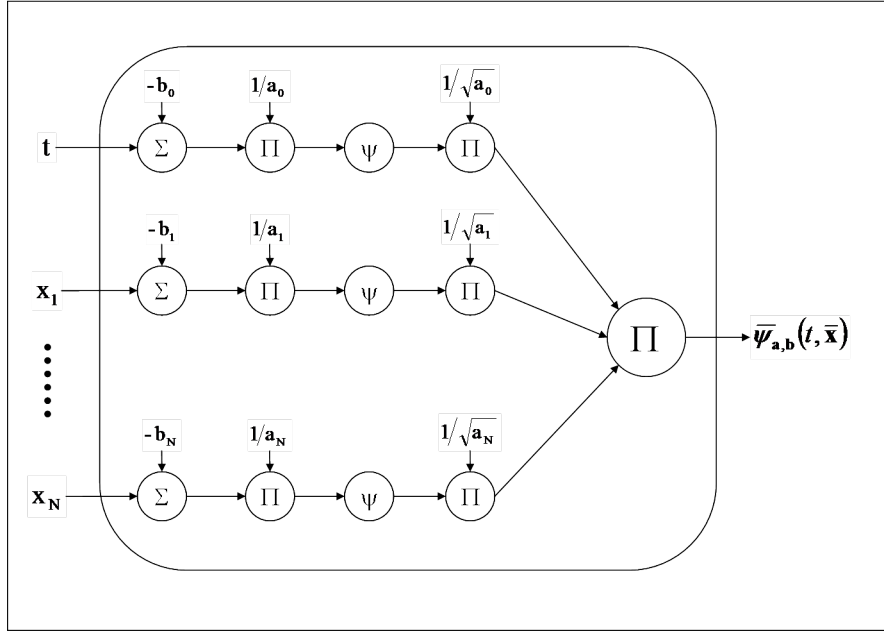


Figure 70: Schematic of a Wavelon with Mother Wavelet ψ

These new ANN's, which build on the theory of wavelet decomposition, are termed Wavelet Neural Networks (WNN) (or Wavelet Networks, or Wavenets). Considering the wavelet decomposition of Equation 39, wavelons such as that described in Figure 70 can be used as neurons in the hidden layer of the neural network in order to produce an approximation \hat{y} , as illustrated in Figure 92. As one can see on the architecture diagram, the resulting neural network is a feedforward neural network.

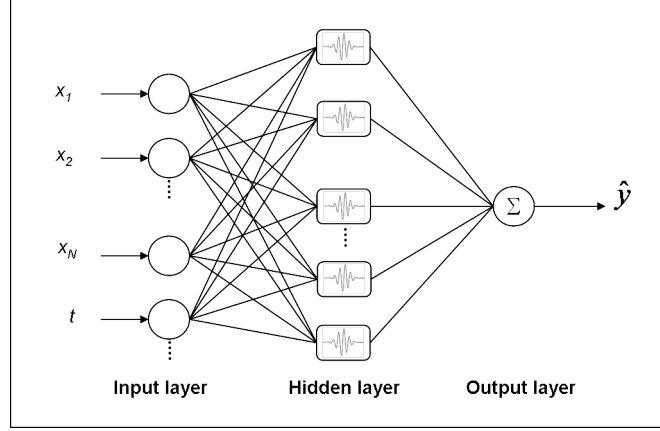


Figure 71: Wavenet Architecture

Originally, wavenets were developed for static function approximation [160, 71]. But they were quickly generalized to the wider field of nonlinear system identification [99, 161, 139, 77, 17]. Practical applications include the modeling of vehicle dynamics [122], time-series forecasting for financial prediction [141], and power quality disturbance detection [67].

Depending on what parameters are fixed before training, WNN's can be classified in two categories: fixed grid WNN's and adaptive WNN's [17]. In the fixed grid WNN, which stems from the DWT of signals, the dilation and translation parameters are fixed in advance, and do not vary during training. On the contrary, the adaptive WNN's stem from the CWT, and the dilation and translation parameters vary during training and make take continuous values.

Training a wavenet is very similar to training an ANN. Given a set of training data, the learning process of a wavenet will adjust, for all the wavelons, the weight w as well as, in case of an adaptive wavenets, the values of the dilation and translation parameters a and b , in order to minimize the regression error on the training data [161, 162, 160].

As shown in Judisty et al., wavenets perform very well when identifying nonlinear dynamic systems, especially for mono-dimensional systems (where the system characteristics are fixed and the experimental dynamic response is identified) [77]. However, in order to capture the time-dependencies of dynamic signals, they sometimes need the addition of feedback loops from the output wavelons to the input wavelons, thus transforming them in a particular case of RNN.

When in their feedforward form, the backpropagation algorithm used in sigmoid based feedforward ANN's can be applied, and wavenets are quick to train. In the presence of feedback loops, the learning time is penalized as the challenges of RNN's arise here as well. Training of the WNN's also becomes more difficult when the number of input variables increases, as explained in Billings [17] and in Kugarajah and Zhang [88]. Therefore, the training speed of a WNN can vary from excellent to poor, depending on the presence of feedback and on the number of variables. In the benchmarking activity, it will then be rated as fair.

If a priori knowledge about the system exists, this knowledge can be easily captured by adding neurons with transfer functions reflecting the insight on the system's behavior. For instance, if it is known that the output behavior of the system contains a sine wave, one can add to the wavenet architecture a neuron with the sine function as the transfer function.

One of the most valuable advantages offered by wavenets is their natural ability to capture the multiscale aspect of dynamic signal. Thus, the wavelet decomposition obtained as an output of the wavenet has a physical meaning that can help analyze and interpret the signal.

The characteristics of wavenets are summarized in Table 8 below.

Table 8: Wavenet Assessment

Criteria	Wavenets
1. Universal Nonlinear Approximator	
2. Multiscale capturing	
3. Training speed	
4. Algorithm complexity	
5. Adaptability to grey-box formulation	
6. Surrogate simulation speed	

Excellent

Very Good

Good

Fair

Poor

3.2.5 Benchmarking Summary and Alternative System Identification Formulation

The assessments of each of the potential nonlinear system identification approaches with respect to the benchmarking criteria set at the beginning of the chapter are recapitulated in Table 9.

Table 9: Benchmarking Summary

Criteria	FFNN	RNN	MDNN	WNN
1. Universal Nonlinear Approximator				
2. Multiscale capturing				
3. Training speed				
4. Algorithm complexity				
5. Adaptability to grey-box formulation				
6. Surrogate simulation speed				

Excellent

Very Good

Good

Fair

Poor

Because wavenets can either be structured as feedforward neural networks or recurrent neural networks, they essentially rate as well as recurrent neural networks in terms of approximation capabilities, and theoretically exhibit better learning and computational efficiency metrics, on par with those of Meijer’s dynamic neural networks. However, training wavenets can become computationally cumbersome when increasing the number of dimensions (i.e. the number of design variables x_i). But it is

in their ability to capture the multiscale dynamics of signals that wavenets fare much better than their competitors. For these reasons, wavenets were selected as the most promising nonlinear system identification approach for the fulfillment of the research objectives of this thesis.

Hypothesis 3 (H3): A nonlinear system identification approach based on wavelet neural networks will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems.

3.3 Contingency Plan: Formulation of an Alternate Hypothesis

After benchmarking the nonlinear system identification approaches, wavenets were chosen as the most promising techniques for the generation of dynamic surrogate models. However, training problems arose when increasing the number of dimensions [17]. Therefore, a contingency approach will be defined in this section.

The basis for the formulation of this alternate approach stems from a reformulation of the verification of transient dynamic constraints. Indeed, instead of verifying that the signal remains within the required dynamic constraints, it is sufficient to verify that the envelope of the signal is contained within these dynamic constraints. The envelope is constituted by two signals, hereafter called the “Sup Envelope” and the “Inf Envelope”, defined such that all points of the signals are comprised between them. The concept of envelope is illustrated in Figure 72.

In this alternate approach, only the envelopes of the signal are generated for each case generated by the Monte-Carlo Simulation of the Time-Domain Filtered-Monte-Carlo. Thus, time-domain overlay cells of the interactive visualization environment (VisTRE) will only plot the envelopes of the dynamic responses, for each design case.

Thus, the new approach is equivalent to applying dynamic transient constraints on a new dynamic system, composed of the original system and a hypothetical “filter”

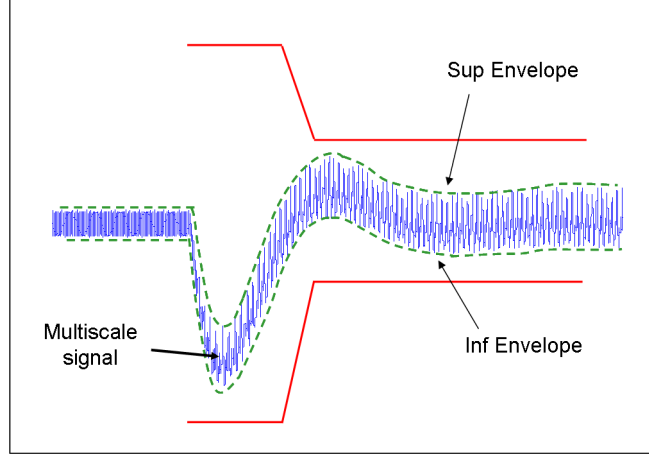


Figure 72: Signal Envelope

that returns the envelope of the signal. The envelopes of signals vary with time, and thus are signals themselves. In order to be computed, they need time-domain simulation coupled with an envelope detection scheme. This is illustrated in Figure 73.

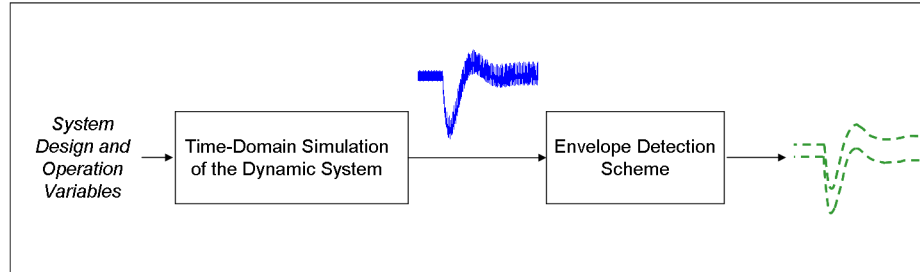


Figure 73: Equivalent System Model for Envelope Detection

Therefore, the limitations of time-domain simulation still hinder the implementation of the Time-Domain Filtered-Monte-Carlo, and dynamic surrogate models of the envelopes will be needed to circumvent these difficulties. In a nutshell, the alternate approach based on envelopes rather than the original time-domain responses is compatible with **Hypothesis 1** and **Hypothesis 2**. By analogy with system identification, the generation of dynamic surrogate models of the envelopes will be called “envelope identification”.

The advantage of dealing with envelopes, which are “smoother” signals, resides in the fact that the high frequency content, corresponding to small time scales, is removed from the original signal. Thus, the nonlinear envelope identification approach does not need to be able to handle multiscale signals. From the benchmarking summary table (Table 9), the corresponding criterion can be disregarded, and feedforward sigmoid-based neural networks seem to be the most promising approach for envelope identification. This can be formulated as an alternate hypothesis, **Hypothesis 3b**, while **Hypothesis 3** will be renamed **Hypothesis 3a**. The two “competing” hypotheses are listed below.

Hypothesis 3a (H3a): A nonlinear system identification approach based on wavelet neural networks will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems.

Hypothesis 3b (H3b): A nonlinear system identification on the envelope of the signals, using sigmoid-based feedforward neural networks, will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems, and the implementation of the Time-Domain Filtered-Monte-Carlo.

3.4 Summary

In this chapter, a benchmarking was performed in order to select an appropriate system identification method for the generation of dynamic surrogate models of dynamic systems under transient dynamic constraints. Wavenet Neural Networks (or wavenets) were chosen as the most promising nonlinear system identification technique, for their documented ability to capture the multiscale aspect of time-domain signals. However, feedforward wavenets may fall in the “curse of dimensionality” and become difficult to train as the number of design variables increases. Therefore, in this thesis, an alternative approach will be taken, where transient constraints will be

applied to the envelope of the system's transient response. Because the multiscale content has been removed, the generation of dynamic surrogate models of the envelopes, termed "envelope identification", will be performed with sigmoid-based feedforward neural networks. The two concurrent approaches are formulated in the hypotheses below:

Hypothesis 3a (H3a): A nonlinear system identification approach based on wavelet neural networks will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems.

Hypothesis 3b (H3b): A nonlinear system identification on the envelope of the signals, using sigmoid-based feedforward neural networks, will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems, and the implementation of the Time-Domain Filtered-Monte-Carlo.

In the alternate approach corresponding to H3b, the envelope of the transient dynamic response of the system has to be obtained. Therefore, an envelope detection scheme needs to be added at the output of the Time-Domain Simulation model of the system. The next chapter will briefly treat envelope detection methods for dynamic signals exhibiting multiscale properties.

Chapter IV

ENVELOPE DETECTION METHODS FOR ENVELOPE IDENTIFICATION

In the previous chapter, an alternate approach was formulated in which the dynamic response of interest is the envelope of the original dynamic response. In this formulated approach, dynamic surrogate models are created for the envelopes instead of the original response: envelope identification is performed. Thus, given a candidate architecture for the dynamic system and its Time-Domain Simulation model, an envelope detection scheme is needed, in order to compute the envelope of the dynamic response produced by the original TDS model. This chapter gives a brief overview of possible envelope detection methods that can be used.

The problem addressed in this chapter, illustrated in Figure 74, can be summarized as selecting an appropriate method for the detection of the envelope signals (sup-envelope and inf-envelope), given an input signal that is highly multiscale. It should be noted that because the input signal is the output of a Time-Domain Simulation activity, it is composed of a finite set of points, thus forming a finite time series $(y_{t_1}, y_{t_2}, \dots, y_{t_n})$, also noted (y_1, y_2, \dots, y_n) . Moreover, the methods described in this chapter will focus on the extraction of the sup-envelope, since the inf-envelope can be obtained from the sup-envelope of the original signal multiplied by -1 .

The detection (or extraction) of the envelope of a signal is a core activity in the processing of amplitude modulated signals. Modulation is a process used in telecommunications, whereby a signal that contains information is altered in order to be more easily transmitted. Demodulation is the inverse process, which consists

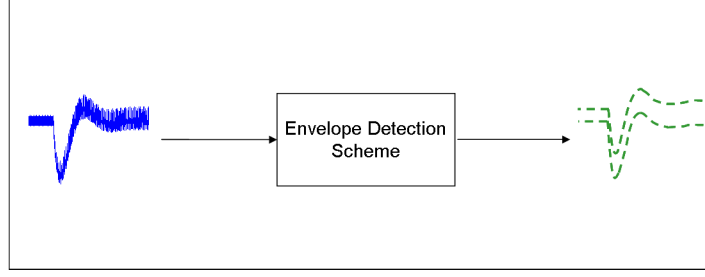


Figure 74: Schematics of an Envelope Detection Scheme

in extracting the information signal from the received signal. Modulating signals is commonly used in the transmission of Radio Frequency (RF) signals (see Kundert [90]).

One popular kind of modulation is the Amplitude Modulation (AM) process. In AM, the signal representing the information to be transmitted, $m(t)$, is multiplied by a “carrier” signal $c(t)$, which is generally a sine wave of high frequency, as described in Equations 27 and 28 :

$$y(t) = m(t) \cdot c(t) \quad (27)$$

$$= C \cdot m(t) \cdot \sin(\omega_c \cdot t + \phi_c) \quad (28)$$

where C is a constant, the amplitude of the carrier

ω_c is the angular frequency of the carrier

ϕ_c is the phase of the carrier

In the case of an amplitude modulated signal with a sine wave carrier, the information signal is the envelope (provided the carrier frequency is high enough) of the modulated signal, divided by the constant C . This is illustrated in Figure 75. Thus, the extraction of the information signal (demodulation) consists in detecting the envelope of the modulated signal.

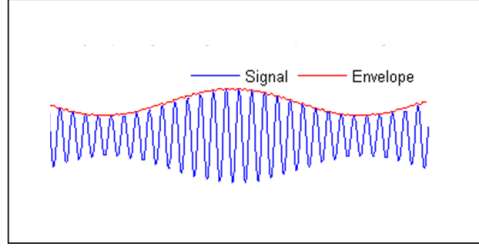


Figure 75: Amplitude Modulated Signal with a Sine Wave Carrier

4.1 *Simple Methods for Envelope Detection*

In this section, simple methods for envelope detection are described. First, a method involving peak detection is described, which will be called the “sliding window method”. Then, a more sophisticated method, based on the Hilbert Transform, is introduced.

4.1.1 Sliding Window

The first method for envelope detection described here is the sliding window method. The idea is to divide the signal according to $n_{\Delta t}$ time windows of a certain time length Δt . For each window, the maximum value taken by the signal is found and kept as a point of the envelope. The envelope is then obtained by linear interpolation between these $n_{\Delta t}$ points. This is illustrated in Figure 76.

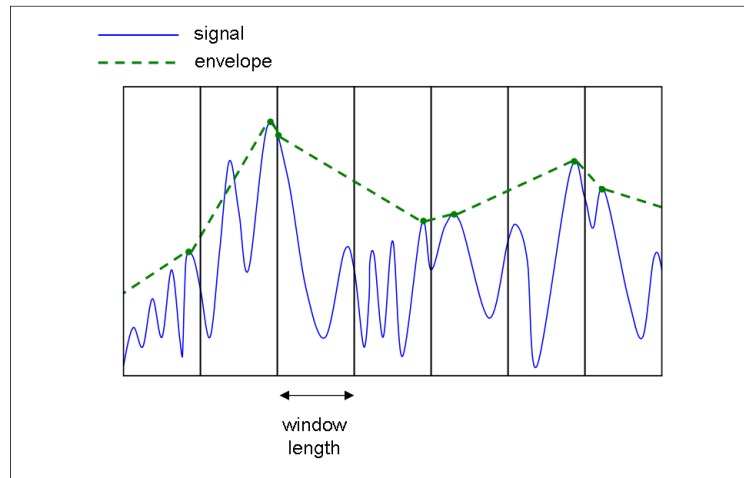


Figure 76: Sliding Window Method

The envelope obtained by the sliding window process can be smoothed by performing a polynomial interpolation instead of a linear one, or by performing a moving average on the resulting envelope, i.e. by taking the average of the envelope, at each point in time, over its contiguous region. This smooths the effect of any abrupt change, which is not desirable for the study of transient regimes.

Also, one can see on Figure 76 that the envelope obtained by the sliding window method does not guaranty that all points of the signal remain below the envelope. However, when the window length is adjusted, one can see, as illustrated in Figure 77, that the computed envelope can represent the behavior of the signal more closely.

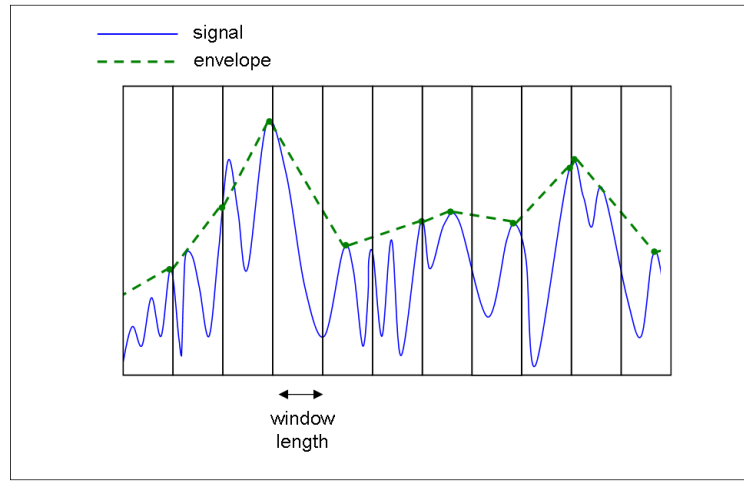


Figure 77: Sliding Window Method with Smaller Window Size

As was hinted at in the previous paragraphs, the accuracy depends on the rate of variation of the signal. If the latter is high, then the window length needs to be small in order for the envelope to carry any physical significance. However, if the window length is too small, the envelope will fit the signal more and more, to the point that it might lose its purpose of filtering out the small time scales. This is a particular challenge for the envelope detection of transient responses since these typically exhibit high rate of variation around the transient perturbation time, thus requiring small window length for the sliding window to be applied effectively.

4.1.2 Hilbert Transform Based Envelope Detection

A more sophisticated and widely used method for the extraction of envelopes of amplitude modulated signals is the envelop detection method based on the Hilbert transform. An overview of the method can be found in Ktonas and Papp [87]. As its name implies, the method relies on the Hilbert transform of a signal. The Hilbert transform $f_1(t)$ of a function $f(t)$ is given by Equation 29:

$$f_1(t) = -\frac{1}{\pi} PV \int_{-\infty}^{+\infty} \frac{f(x)}{t-x} dx \quad (29)$$

where the notation $PV \int$ refers to the Cauchy principal value :

$$PV \int_{-\infty}^{+\infty} f(x) dx = \lim_{R \rightarrow +\infty} \int_{-R}^{+R} f(x) dx \quad (30)$$

The envelope $m(t)$ of a signal can be given analytically by Equation 31:

$$m(t) = \sqrt{f(t)^2 + f_1(t)^2} \quad (31)$$

The envelope detection method using the Hilbert transform is very convenient because it gives an analytical solution. Also, the Hilbert transform, which in fact shifts the signal by a phase of 90° , is generally computationally efficient to calculate. However, this method is only accurate for bandpass signals, i.e. for signals with frequency spectra that are compactly supported (zero outside a finite interval), with supports that do not include zero. Figure 78 shows an example of the frequency spectrum of a bandpass signal. This condition means that the Hilbert transform envelope detection method is not suitable for signals that contain a DC component (low frequencies).

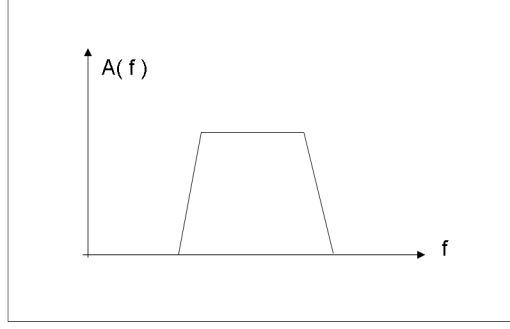


Figure 78: Frequency Spectrum of a Bandpass Signal

4.2 *Envelope Detection Method Based on Multiresolution Analysis with Wavelets*

In the previous sections, two simple methods were introduced: the sliding window method and the Hilbert transform based method. It was seen that the multiscale aspect of the signal and the potential high rate of variation induced by transient perturbations pose a challenge to the application of those methods. Because wavelets allow to deal with multiscale signals and separate the scales, they have been increasingly used within envelope detection methods, as shown in Sheen and Hung [134].

4.2.1 Multiresolution Analysis

By using a Multiresolution Analysis (MRA), a process that was briefly introduced in Chapter III, the scales of a multiscale signal can be separated. The MRA process based on wavelets will be explained in more detail here. As seen in Mallat [103], discrete lattices of wavelets can be used to study the signal at different scales. Indeed, if one takes a mother wavelet ψ , the family formed by the set defined in Equation 32 is a complete orthonormal basis of $\mathcal{L}^2(\mathbb{R})$, provided the mother wavelet verifies certain conditions. In this case, the family of daughter wavelets are said to offer a multiscale resolution.

$$\left\{ \psi_{j,k}(t) = \frac{1}{\sqrt{2^j}} \psi\left(\frac{t - k \cdot 2^j}{2^j}\right) : i, j \in \mathbb{Z} \right\} \quad (32)$$

In the notation given above, the scaling factor is $s = 2^j$ and the translation factor is $\tau = k \cdot 2^j$. The relationship between the scaling factor and the translation factor shows that their discretization forms a kind of lattice of allowable values that is referred to as a “dyadic grid”. The dyadic grid, representing the values taken by the scaling and translation factor is given in Figure 79. As one can see, as the s decreases (j decreases), the resolution obtained from the wavelet decomposition increases.

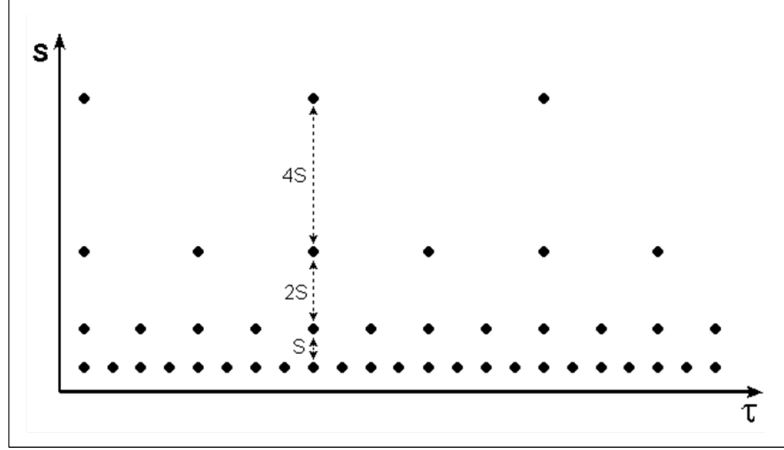


Figure 79: Dyadic Grid of Scaling and Translation Factors in a MRA

The orthonormality of the MRA implies that the discrete wavelet decomposition in this basis is unique and is given by Equation 33:

$$y(t) = \sum_{j \in \mathbb{Z}} \sum_{k \in \mathbb{Z}} \langle y, \psi_{j,k} \rangle \cdot \psi_{j,k}(t) \quad (33)$$

The coefficient of the wavelet decomposition expressed above is theoretically obtained with an integral. The associated numerical complexity can be avoided by the introduction of an auxiliary function, called “*scaling* function”, that exhibits certain properties. For a given MRA associated with a mother wavelet ψ , there exists a *unique* scaling function φ [103, 35, 69, 73]. Given a scaling function φ , daughter scaling functions are defined in the same fashion as the daughter wavelet functions were in Equation 32.

In literature, there are many options for the selection of appropriate mother wavelets for MRA. One of the most used sets of wavelets is the set of functions that were constructed by Daubechies [35]. The Daubechies set of wavelets, called Daubechies wavelets, was constructed recursively, and the couple of scaling and mother wavelet functions for the fourth order (noted “db4”) is plotted in Figure 80.

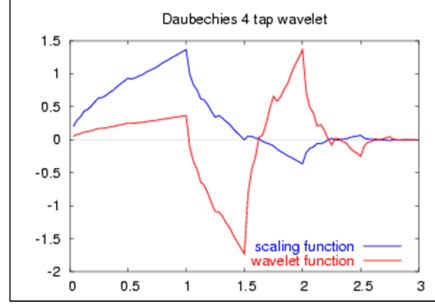


Figure 80: Daubechies Wavelet and Scaling Functions at the Order 4 [105]

The scaling functions complement the mother wavelets in the wavelet decomposition, so that for a given desired resolution J , the function f can be expressed as in Equation 34.

$$f(x) = \sum_{k \in \mathbb{Z}} c_{J,k} \cdot \varphi_{J,k}(x) + \sum_{j \geq J} \sum_{k \in \mathbb{Z}} d_{j,k} \cdot \psi_{j,k}(x) \quad (34)$$

From the equation above, one can see that the first term, related to the scaling function, contains information on the function at the resolution J , while the second term contains information on the finer scales $j \geq J$. Thus, the first term can be taken as the approximation of f at the resolution level J and the second term can be considered as the detail of the function f for higher resolutions. The approximation at level J and the detail at level J are noted $A_J(f)$ and $D_J(f)$.

An essential property of the combination of scaling and wavelet functions is that they exhibit “self-similarity” laws, as expressed in Equations 35 and 36 for levels j and $j + 1$:

$$\varphi(2^j t) = \sum_k h_{j+1}(k) \cdot \varphi(2^{j+1} t - k) \quad (35)$$

$$\psi(2^j t) = \sum_k g_{j+1}(k) \cdot \varphi(2^{j+1} t - k) \quad (36)$$

These self-similarity laws imply that the decomposition at any level can be recursively computed from the decompositions at higher levels J , as seen in Equation 37.

$$A_J = A_{J+1} + D_{J+1} \quad (37)$$

This means that the approximation at the resolution $J + 1$ is obtained by the subtracting the details, with high frequency components, to the approximation at higher resolution J . This is represented in the MRA decomposition tree of Figure 81.

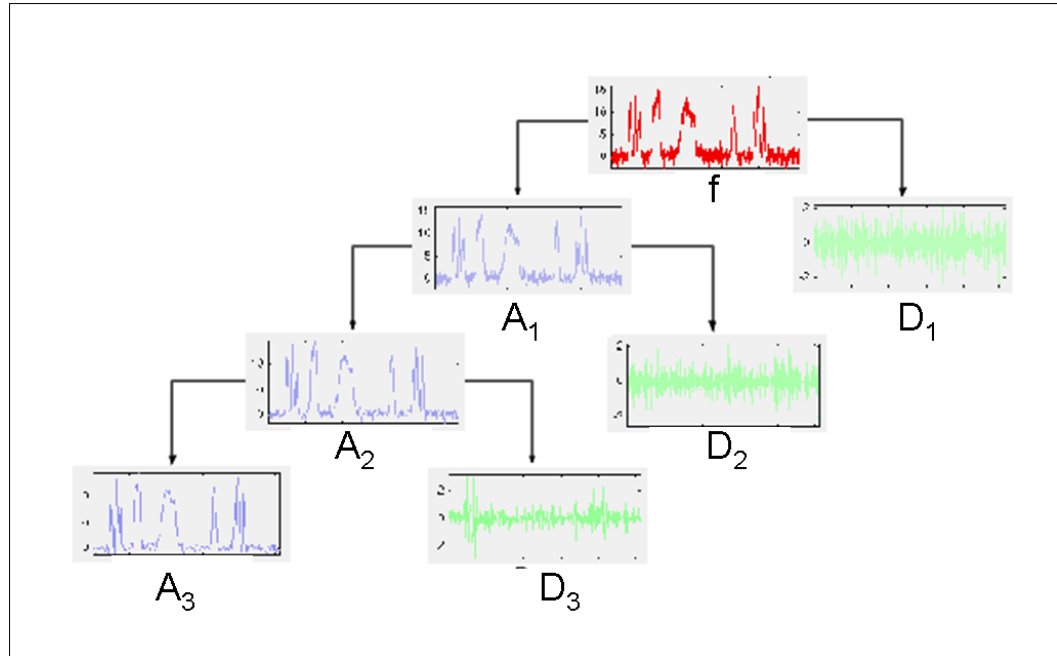


Figure 81: Multiresolution Analysis Decomposition Tree [105]

The process that computes the approximation and detail at level J can be obtained in a computationally efficient manner by applying filters to the approximation at level

$J - 1$: a lowpass filter (keeping the low frequencies) for the computation of A_J and a highpass filter (keeping the high frequencies) for the computation of D_J . When dealing with discrete signals, such as the output of a Time-Domain Simulation, the outcome of each filtering process is downsampled by a factor 2, which is justified by the fact that each level increment corresponds to going up the scales of the dyadic grid of Figure 79, with half the number of time samples. The whole process constitutes the basis of the Fast Wavelet Transform (FWT) [35, 103].

4.2.2 Envelope Detection Method Based on MRA Decompositions

It was seen that by applying a MRA, scales can be segregated and noisy signals can be separated into a global trend (the result of the approximation A_J at a certain level J) and the noise (the sum of Dj for all levels $j \leq J$), centered around zero. This decomposition is illustrated in Figure 82.

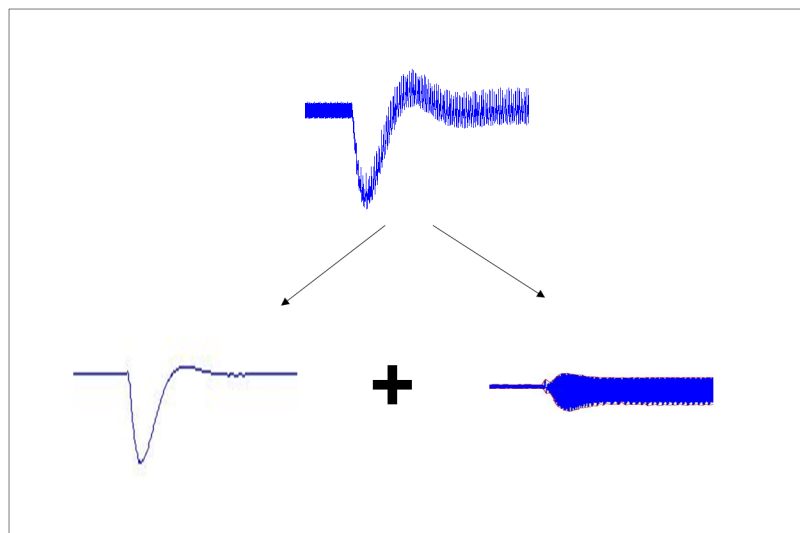


Figure 82: Signal Denoising

As one can see from the figure, the noise signal, corresponding to the ripple around the global trend and resulting from the aggregation of details, has envelopes that present lower rates of variation. Therefore, the sliding window method can be applied with more precision to the ripple signal.

Therefore, the envelope (sup or inf) of the signal will be obtained by summing the trend and the corresponding envelope of the ripple signal. The overall envelope detection method is summarized in Figure 83.

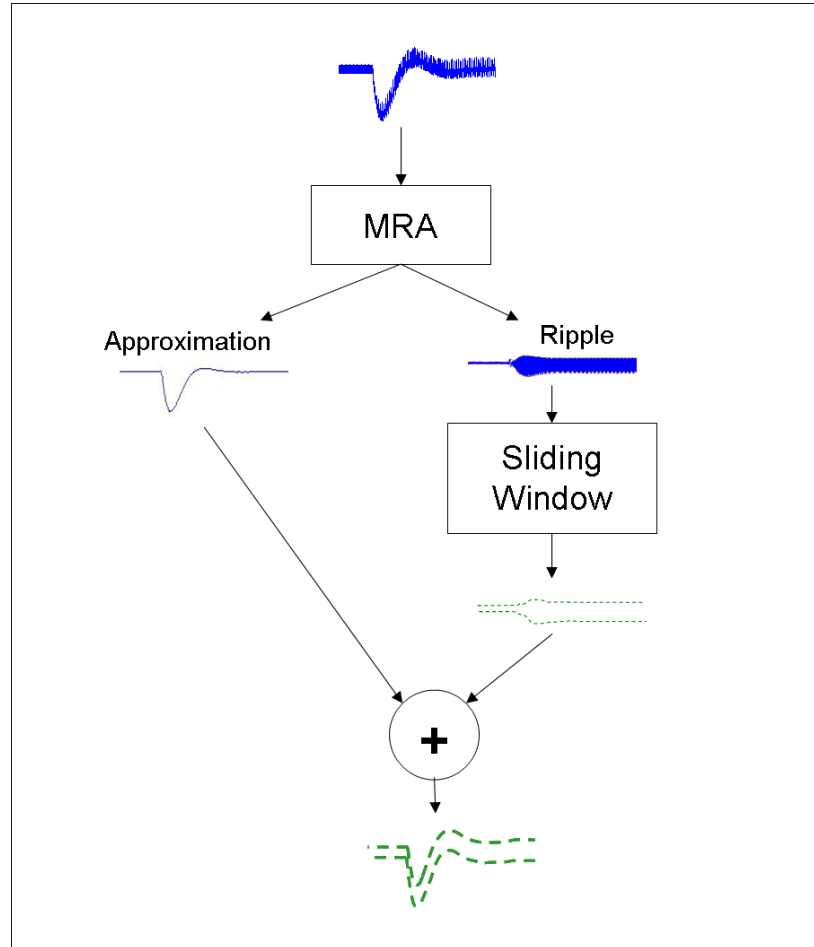


Figure 83: Envelope Detection Method Based on MRA

4.3 Summary: MRA-Based Envelope Detection Method for Envelope Identification

In this chapter, envelope detection methods were discussed. The “traditional” methods that were discussed are the sliding window method (or “windowing method”), and the Hilbert transform method.

Then the envelope detection method used in this thesis was introduced: it consists of using a wavelet-based multiresolution analysis (MRA) to separate the time-domain signal into an “approximation signal”, which gives the global trend of the transient response, and a ripple signal, corresponding to the aggregation of all the “detail signals” obtained in the course of the MRA. The envelope of the original signal is the sum of the approximation and the envelope of the ripple signal. The latter is obtained by the application of a sliding window method.

In the alternate approach of this thesis, resulting from the alternate Hypothesis 3b, sigmoid-based feedforward neural networks are generated in order to approximate the envelope of the signal, for which transient dynamic constraints are then evaluated. There are potentially several options for the application of neural networks. Neural networks can be trained in order to fit the envelope of the original signal, output of the summation in Figure 83. However, since in the process, the approximation (at a certain resolution level) is generated, one might as well keep this information for the subsequent transient response analysis activity in VisTRE. Thus, a first neural network will be trained to fit the approximation (trend signal). Moreover, the envelopes of the ripple signal typically exhibit less rate of variation, therefore presenting the advantage of being easier to fit. Therefore, two neural networks will be trained to fit the sup-envelope of the ripple, and its inf-envelope. Thus, for each dynamic response of interest (e.g. voltage, current, etc.), three dynamic neural networks will be generated, as illustrated in Figure 84, which shows how the envelope detection method is used in the context of envelope identification.

Naturally, the envelope of the original signal will be obtained by summation of the output of the neural network for the approximation signal and the outputs of the neural networks for the ripple envelopes. The process is illustrated in Figure 85.

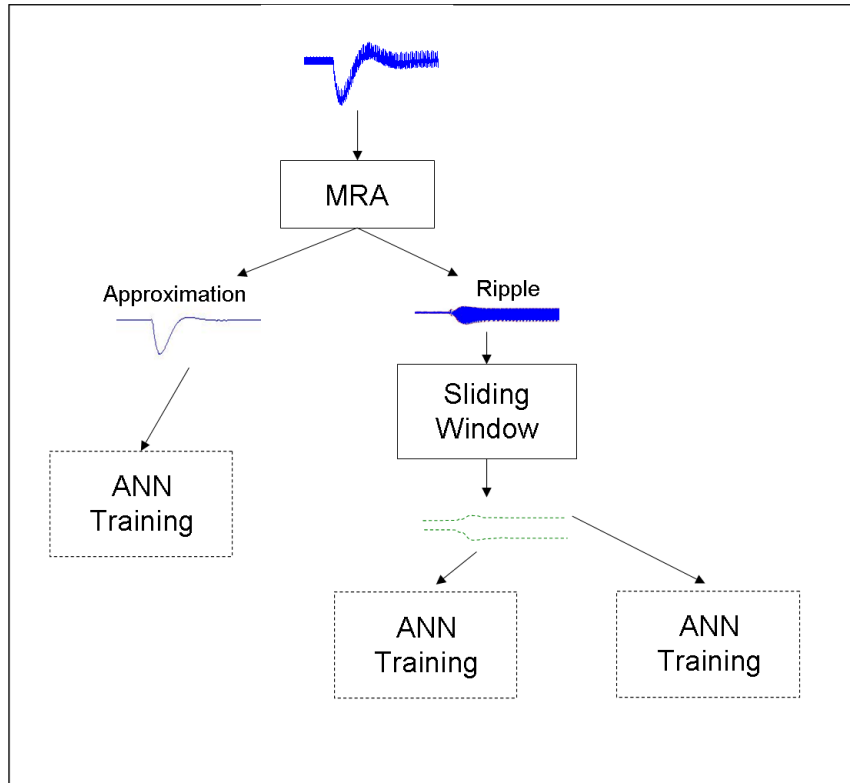


Figure 84: Envelope Identification Process

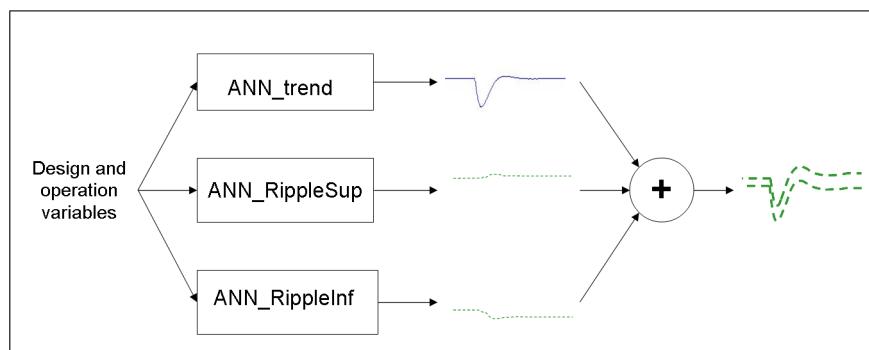


Figure 85: Envelope Reconstruction with Dynamic Surrogate Models

Chapter V

HYPOTHESES AND METHODOLOGY FORMULATION

The objective of this thesis is to develop a methodology that facilitates the integration of transient related constraints earlier into the design specification of aircraft dynamic systems. Several challenges were identified, and these challenges led to the formulation of research questions that allow the decomposition of the problem into manageable ones. The investigation of the research questions in turn serves as sub-level objectives for this thesis. Based on observations and a literature survey on the topic of system identification, a set of hypotheses has been formulated as answers to the research questions. These hypotheses will translate into a design methodology, parts of which have already been described in the previous chapters.

5.1 Recapitulation

Before the proposed methodology is described, the research questions and the hypotheses are restated below, as a way to clearly establish the thought process that led to its formulation.

Research Objective 1: Develop a methodology that integrates transient regime analysis and pertaining dynamic constraints into the design synthesis of aircraft dynamic systems.

Research Question 1 (RQ1): How can one efficiently and thoroughly explore the design and operation spaces while accounting for dynamic responses and pertaining uncertain dynamic constraints?

Hypothesis 1 (H1): A data farming approach, based on the integration of time as a dimension in the the Filtered-Monte-Carlo approach, will conduce to the efficient and thorough exploration of the design/operation space for dynamic signals with dynamic constraints.

Research Question 2 (RQ2): How can one make the system simulation process more efficient in order to speed up the optimization/verification of dynamic systems and facilitate the implementation of the proposed Time-Domain Filtered Monte-Carlo technique?

Hypothesis 2 (H2): Dynamic surrogate modeling of time domain simulation models will enable the efficient implementation of the Time-Domain Filtered Monte-Carlo technique for the exploration of design/operation space of dynamic systems subject to dynamic constraints.

Research Question 3 (RQ3): What system identification approach is suitable for the generation of dynamic surrogate models in the context of transient time domain simulation of dynamic systems subject to dynamic constraints?

Hypothesis 3a (H3a): A nonlinear system identification approach based on wavelet neural networks will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems.

Hypothesis 3b (H3b): A nonlinear system identification on the envelope of the signals, using sigmoid-based feedforward neural networks, will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems, and the implementation of the Time-Domain Filtered-Monte-Carlo.

5.2 Methodology

The hypotheses formulated in the previous section can be transposed into a methodology for the design of aircraft dynamic systems subject to transient-related dynamic constraints. This methodology consists of several steps that are detailed in the following sections. Because an alternate approach, based on the analysis of the envelopes of the dynamic response, was formulated, the methodology formulated in this thesis consists of two competing approaches, and some of the steps outlined hereafter will be subdivided into competing sub-steps.

5.2.1 Step 1: Define the Problem

First, it is crucial to identify the parameters that will be considered during the design of the dynamic system. Most of these parameters will be directly derived from the requirements issued by the earlier design phases. These parameters can be grouped into two categories: the responses and the input variables.

The responses are the parameters that are being optimized or that are under constraints. For each response, the designer has to determine whether they are static or dynamic, and determine the constraints that regulate them.

The input variables are the controllable parameters that have an impact on the responses. For each response, the design team has to ask the following questions: “*what are the factors influencing the response?*” and “*what are the ranges of variation for these input variables?*”. For the dynamic responses, it is essential to determine the operation scenarios that might trigger transient behaviors, and determine the input variables that have an influence on the shape of these transient behaviors.

5.2.2 Step 2: Create the Modeling and Simulation Environment

The next step consists in setting up a M&S environment. From the functional and physical specifications passed from the previous design phases, and in light of the parameters and constraints identified in Step 1, the design team now creates models

capturing the behavior of the responses with respect to the input variables. As seen in the first and second chapters, if the dynamic responses are subject to transient-related dynamic constraints, a time-domain simulation (TDS) environment has to be created in order to compute their time-domain behavior.

In general, at this phase, the design team has identified a physical decomposition that can potentially meet the design requirements. The system is thus composed of elementary components, for which simulation models are often already available in popular TDS software packages such as Simulink [106] or Dymola [45].

As seen in Chapter II, setting up a TDS framework requires the designer to make choices regarding the simulation time step and time window. The designer has to choose between a fixed-increment time advance mechanism or a next-event one: whether the time step will be fixed or varied in an adaptive manner over the course of the simulation. This has a direct impact on simulation run time. But because the outputs of dynamic surrogate models are explicit functions of time, the choice of the time-advance mechanism of the original TDS environment has no impact on the simulation run time of the dynamic surrogate models. Thus, for the purpose of generating data for the training of the surrogate models, one might use fixed-increment time on the original TDS tool, without preventing the designer to use the resulting dynamic surrogate model with a next-event type of time advance mechanism.

Because fixed-increment time advance is a particular case of the next-event type, the latter would be preferred. But for the generation of wavenets (subjects of H3a), a fixed-increment time advance mechanism is chosen, as it simplifies the way data is treated during training.

5.2.3 Step 3: Test Planning - Design of Experiments

As explained in the previous chapters, the design and operation space exploration environment will not directly run the time-domain M&S environment created in Step 2. Instead, a dynamic surrogate model of the latter, generated with wavenets (H3a) or sigmoid-based feedforward neural networks on the response envelopes (H3b), will be created. This is done in two phases: first, in a process called “test campaign”, the TDS model is run multiple times in order to compute the behavior of the responses for various design/operation scenarios. The resulting collection of inputs and outputs constitutes the training data. The second phase is the training (or learning), where the neural networks are set up and optimized so that the neural architecture approximates the TDS outputs (or the envelope) for the training data. In Step 3, the design and operation scenarios (or test cases) that will be simulated during the test campaign are determined. The output of this step is thus a test plan.

In system identification, the input variables are often dynamic, and the training data is created by running the TDS model at random input settings: for each case, the dynamic input signals are allowed to fluctuate randomly [148]. However, the inputs here are considered to be static, as explained in Chapter III. Therefore, a more sophisticated approach will be used to create the test plan: Designs of Experiments (DoE). DoE’s are statistical techniques that allow for an efficient sampling of the design and operating space with a minimal number of tests [117]. Because of the way the variable settings are arranged, it is possible to isolate the effects of variables independently from one another. For a given number of input variables, there are several possible designs [59]. The choice of the design depends on the number of runs that can be afforded, and on where in the design and operating space the attention is focused. In the context of neural network training, Latin Hyper Cube (LHC) designs (or other “space-filling” designs) are generally favored, as they efficiently sample the entire input space without creating much bias of the resulting model towards a

particular input variable [159]. LHC's are a generalization of the well-known Latin Squares, in which two cells that belong to the same row or column cannot have the same value. In LHC's, the number of tests to be performed is selected in advance. Figure 86 depicts an example of LHC sampling in the case of two dimensions. In this figure, the points were obtained randomly and rearranged in such a way that the latin square condition (2 points cannot have the same value of x or y) is satisfied. It is a case of Random Latin Hyper Cube (RLHC).

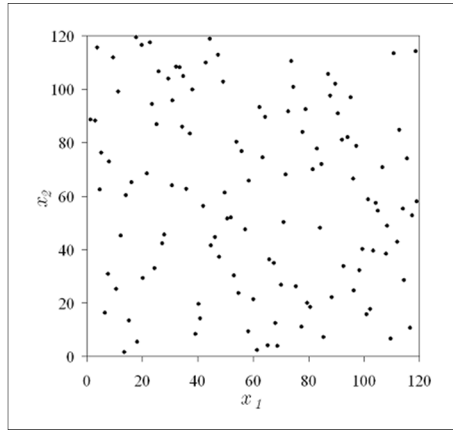


Figure 86: Random Latin Hyper Cube Design for Two Dimensions [12]

In order to sample the design space more thoroughly, LHC designs can be improved by an optimization process [124, 96]. The result is then called Optimal Latin Hyper Cube (OLHC). Given the number of design variables and the number of desired runs m , corresponding to settings of design variables, the algorithm will place the m design points in the design space and maximize the minimum distance between points [159]. This will effectively “spread” the points across the design space. In Bates et al., the example of Figure 86 was optimized using a genetic algorithm. The resulting OLHC is given in Figure 87.

The DoE produces a test table describing the runs that will be performed in order to produce data that will be used subsequently during training, in order to generate dynamic surrogate models. As was seen in the figure depicting the generic process

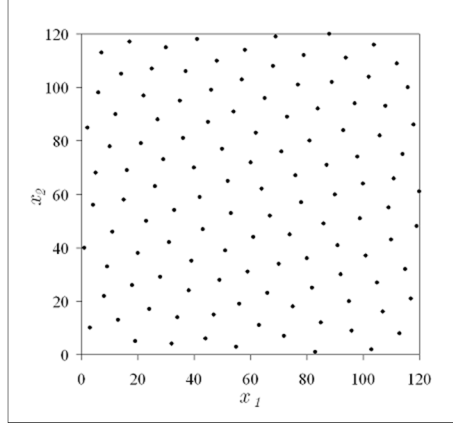


Figure 87: Optimal Latin Hyper Cube Design for Two Dimensions [12]

for a simulation study (Figure 48), a model needs to be validated. Therefore, the test plan generated by this step includes, besides the DoE cases (called training cases), a set of additional cases that will be run in order to obtain data points that will be used to validate the approximation capability of the dynamic surrogate models. These additional cases, called validation cases, are generated randomly.

As mentioned in previous sections, when simulating a dynamic system with a TDS model, an important parameter to consider is the simulation sampling time for the discretization of the problem. This step should define, along with the test plan, the sampling time intervals at which the various design/operation scenarios should be simulated.

5.2.4 Step 4: Test Campaign - Collect the Training Data

In Step 4, the test plan designed in Step 3 is carried out so as to obtain the transient behavior of the signals of interest for each test case, as well as the static responses that will be used to populate the static scatterplot of the Visual Transient Response Explorer (VisTRE), the interactive visualization environment defined in Chapter II.

For the static responses, the output of step 3 is a set of matrices, one for each static response of interest, giving the response value for each case of the test campaign. For each static response R^k (e.g. weight, cost, etc.), the results of the test campaign are

lumped into a vector $(R_1^k, R_2^k, \dots, R_m^k)$, where m is the number of test cases in the test plan generated by the DoE in step 3.

For the dynamic responses, which will populate the innovative time-domain cells of VisTRE, the output of step 3 is a set of matrices, where each matrix contains the time behavior for a particular dynamic response. This is illustrated in Figure 88.

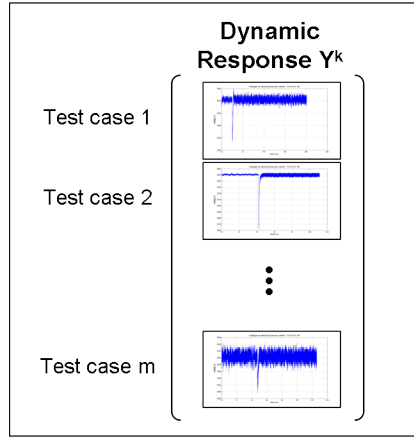


Figure 88: Test Campaign Output for the Dynamic Response Y^k

Now the data needs to be processed and arranged in such a way that it is ready for the subsequent generation of dynamic surrogate models. The process will differ depending on whether the wavenet-based methodology or the envelope-based one is implemented.

5.2.4.1 Data Postprocessing for Wavenet System Identification (H3a Method)

Because the wavenets deal directly with the dynamic outputs of the TDS activity of step 3, little postprocessing is needed. For each dynamic response $y^k(t)$, the time series produced by each test case in the test campaign are lumped into a time-domain training matrix. This is illustrated in Figure 89. One can see that if there is only one time sample simulated during the test campaign, then the time-domain training matrix is reduced to a single column matrix, similar to the training matrix obtained for a static response R described above. This is coherent since a signal simulated for only one time instant should be considered as a static response.

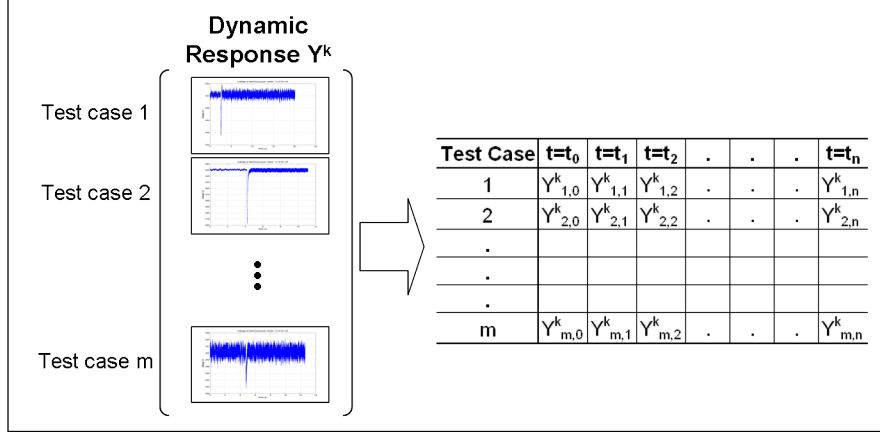


Figure 89: Training Data Collection for the Dynamic Response Y^k

5.2.4.2 Data Postprocessing for Envelope Identification (*H3b Method*)

In the alternate method, the data resulting from the test campaign needs to be processed in order to facilitate the training of the neural networks. Thus, the envelope detection scheme, discussed in Chapter IV and described in Figure 84, needs to be applied to each test case of the test campaign. For each test case, this produces three time series per dynamic response: one for the trend of the signal, one for the sup-envelope of the ripple, and one for the inf-envelope of the ripple. These time series will be the input to the training activity of the sigmoid-based feedforward Artificial Neural Networks (ANN). However, in order to make the training data less heavy to handle, subsampling of the time series (for the trend and ripple envelope) is performed. Because the transient perturbation often induces high rates of variation, the sampling will be finer around the perturbation instant, so that as to introduce a bias in the training of the ANN's. As a result, the ANN's will be able to capture the behavior around the transient perturbation with more accuracy. The concept of adaptive sampling rate is illustrated in Figure 90.

At this stage, for each test case of the test campaign (DoE table and validation cases), this step has produced three time-series (of the trend and the two envelope signals of the ripple) per dynamic responses. These time series need to be aggregated

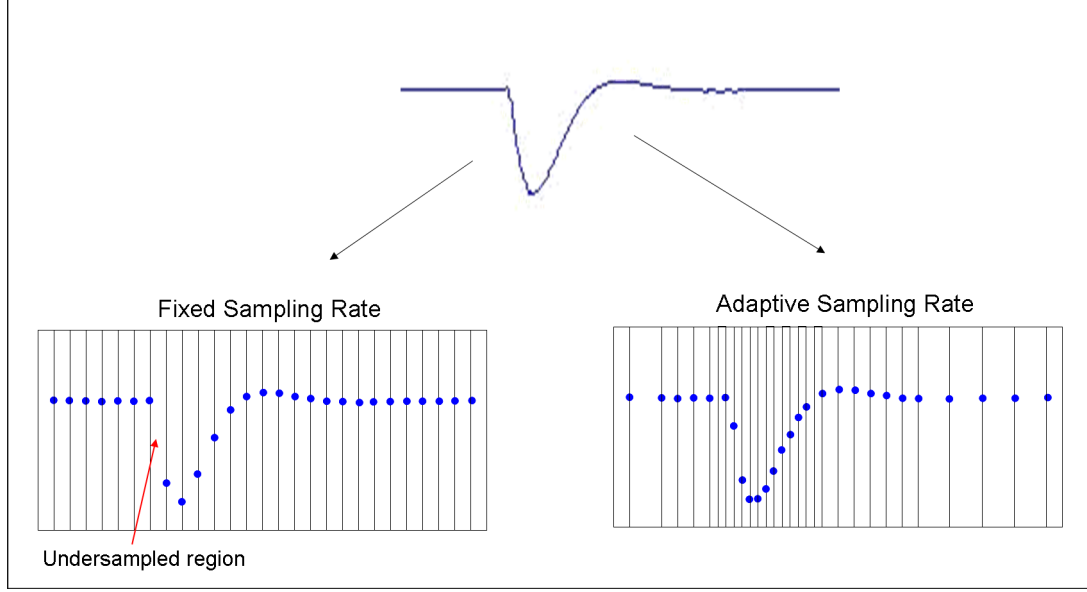


Figure 90: Subsampling of the Trend Signal

and arranged in order to be handled by the neural network training procedures. The outcome of this step is therefore a set of three training matrices per dynamic response: one training matrix for the training of the signal trend neural network (ANN_Trend), one training matrix for the training of the ripple sup-envelope neural network (ANN_RippleSup), and one training matrix for the training of the ripple inf-envelope neural network (ANN_RippleInf).

During training of the sigmoid-based feedforward neural networks, the time parameter t is treated in the same manner as design variables x_i are treated. Therefore, each training case of the training procedure will not be the full (subsampled) output signal of a test case of the test campaign, but a sample at an instant t of these output signals. Therefore, if the number of test cases run in the test campaign of step 4 is m , and if the number of sampled points obtained after subsampling the i^{th} signal is n_i , then the training matrix contains $\sum_{i=1}^m n_i$ rows, corresponding to actual training cases. This is illustrated in Figure 91.

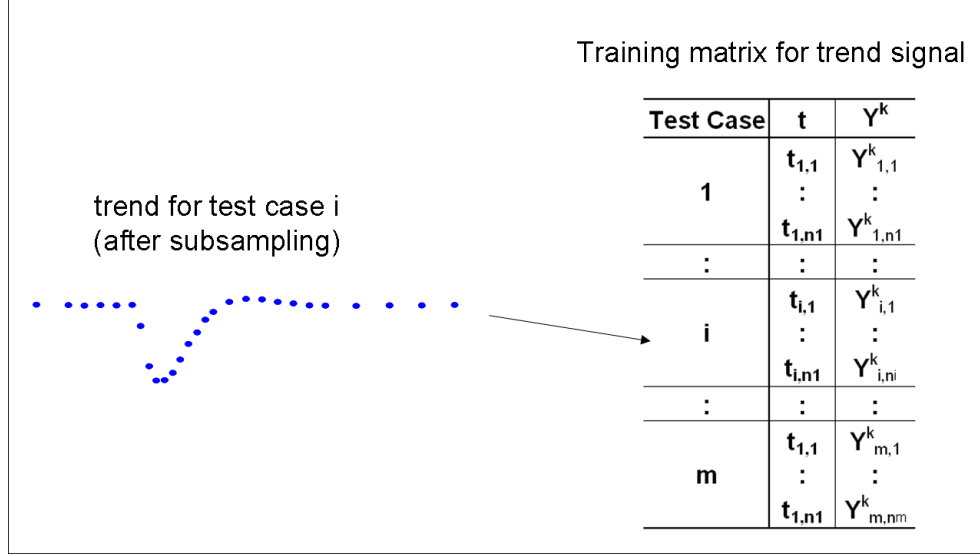


Figure 91: Training Matrix for the Trend Signal of the Dynamic Response Y^k

5.2.5 Step 5: Create Dynamic Surrogate Models - Train the Neural Networks

After the test campaign, the collection of the training data and its preparation, the training (or learning) process of the wavenet (or, for the alternate method, the sigmoid-based feedforward neural networks) can start. The process described in this step applies to one dynamic response y (with samples Y), and should be repeated for each dynamic response y^k . Learning is an optimization process in which an objective function, modeling the total regression error, is minimized. There are multiple ways to formulate the regression error. Introduction of various possible error metrics is given in Sjoberg et al. and in Judistky et al. [139, 77]. As mentioned in Chapter III when discussing the training of feedforward neural networks, popular choices of regression error are based on MSE. This will be the case in this methodology, where the formulation adopted here is adapted from Zhang and Benveniste [162], and given in Equation 38.

$$Err = \sqrt{\sum_{i=1}^m \sum_j \left(Y_{i,j} - \hat{Y}_{i,j} \right)^2} \quad (38)$$

where $Y_{i,j}$ is the training data value of the dynamic signal for the i^{th} test case
at the instant t_j
 $\hat{Y}_{i,j}$ is the regressed output of the wavenet for the i^{th} test case
at the instant t_j

The learning process, which is iterative in nature, is similar for the training of the wavenets on the time-domain signals and for the training of the neural networks on the trend and ripple envelopes of the signal. The training processes for both competing approaches are now described.

5.2.5.1 Training the Wavenets

The input of the training process is the training matrix pictured in Figure 89. The objective of the process is to determine a wavenet architecture that produces an output that best approximates the training matrix. This architecture is defined by the number of wavelons N_w in the hidden layer, the values of the scaling and translation coefficients, and the values of the weights of the connections between wavelons. Recall the equation of the wavelet decomposition, which the wavenet implements:

$$y(\bar{\mathbf{x}}, t) \approx \sum_{k=1}^{N_w} w_{\mathbf{a}_k, \mathbf{b}_k} \bar{\psi}_{\mathbf{a}_k, \mathbf{b}_k}(\bar{\mathbf{x}}, t) \quad (39)$$

where $\bar{\mathbf{x}} = (x_1, \dots, x_N)$ is the vector of design variables

N_w is the number of wavelets in the approximation

$\mathbf{a}_k = (a_{k,0}, a_{k,1}, \dots, a_{k,N})$ are the scaling coefficients of wavelet k

$\mathbf{b}_k = (b_{k,0}, b_{k,1}, \dots, b_{k,N})$ are the translation coefficients of wavelet k

$w_{\mathbf{a}_k, \mathbf{b}_k}$ is the weighting coefficient associated to wavelet k

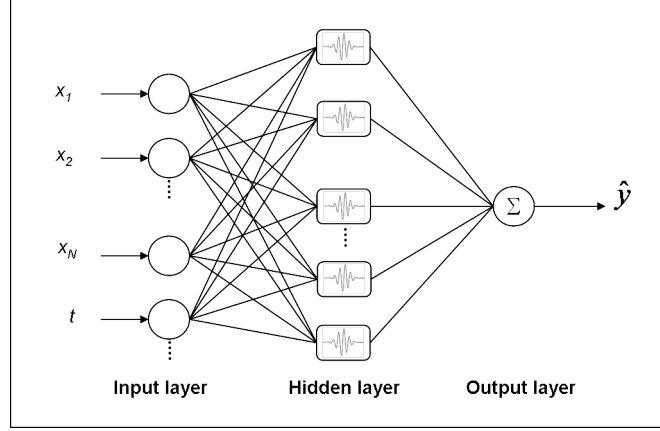


Figure 92: Wavenet Architecture

Each wavelet in the equation above is a multidimensional wavelet, that is modeled by a wavelon in the wavenet architecture, recalled in Figure 92, where the wavelons are represented as the rounded rectangles.

As with any neural network, training the wavenet is an iterative process. First, a wavenet architecture, with a number of wavelons, is assumed. Given this architecture, the wavenet output $\hat{Y}_{i,j}$, and thus the regression error Err , depends on the values of the wavelon connection weights w and the wavelon dilation and translation parameters a and b . During training, these parameters are grouped into a vector $\bar{\theta}$, so that the error Err is now a function of $\bar{\theta}$ only. Learning can now be formulated as an unconstrained optimization process where the vector $\bar{\theta}$ is adjusted in order to minimize the regression error $Err(\bar{\theta})$. Here, a conjugate-gradient optimization algorithm will be used. If the regressive performance of the wavenet is not satisfactory after convergence of the optimization algorithm, the process is reset and repeated with a higher number of wavelons. The overall process for the training of wavenets is summarized in Figure 93. At the outset of the wavenet learning process, the wavenet output neuron provides an approximation of the dynamic signal, thus constituting a dynamic surrogate model of the dynamic system.

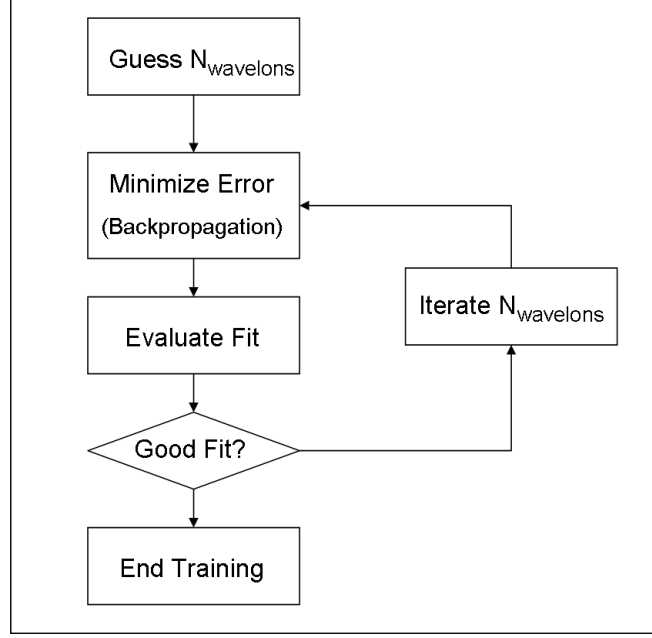


Figure 93: Wavenet Training Process

5.2.5.2 Training the Feedforward Sigmoid Neural Networks

For the alternate approach, derived from Hypothesis 3b, traditional sigmoid-based feedforward neural networks are trained in order to approximate the trend signal of the dynamic response, the sup-envelope of the ripple, and its inf-envelope. Training these feedforward neural networks is a process similar to training wavenets. In Figure 94, the architecture of the neural networks to be trained is recalled.

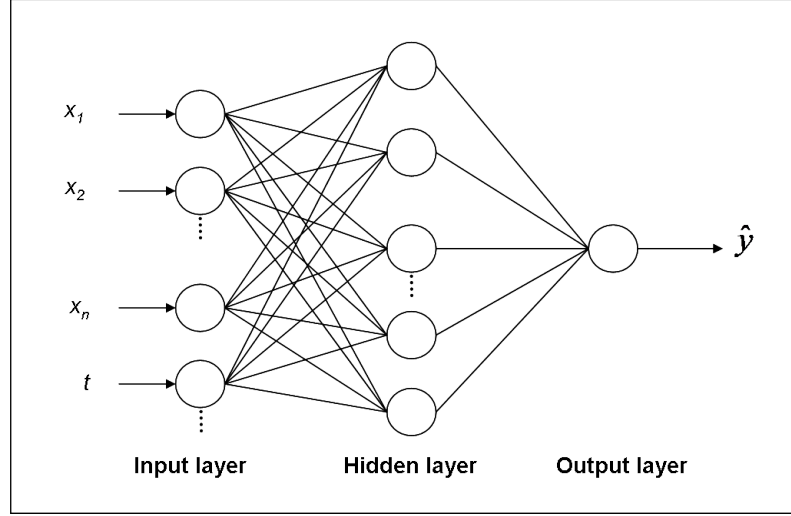


Figure 94: Dynamic Feedforward Neural Network

The input of the training process is a training matrix depicted in Figure 91. The objective of learning is to determine a neural network architecture that minimizes the error for the points corresponding to the training matrix. A neural network architecture is defined by the number of neurons in the hidden layer, and the weights w and b of the connections (or synapses) between neurons (cf. Figure 51 of a neuron in Chapter II for exact descriptions of w and b).

As with the wavenets, first, a number of neurons in the hidden layer is assumed. Then, for this architecture, the synaptic weights w and b are optimized in order to minimize the MSE-based error metric defined in Equation 38. The optimization process is carried out using the widely popular backpropagation algorithm, presented in detail in Appendix B. The overall process for the training of neural networks is summarized in Figure 95.

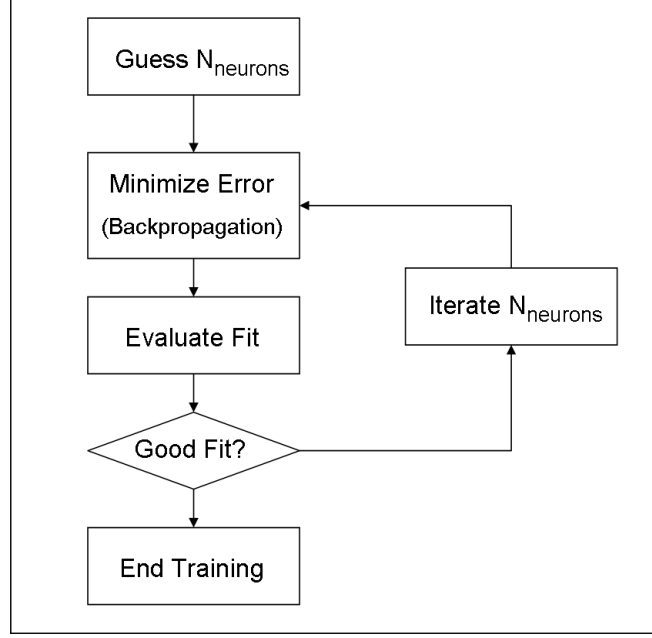


Figure 95: Neural Network Training Process

Finally, the same process is used to create surrogate models on the static responses R^k , which will be used to populate the static scatterplot in VisTRE, the interactive visualization environment of the Time-Domain Filtered-Monte-Carlo approach formulated in Chapter II.

5.2.5.3 Check the Goodness of Fit

Whether dynamic surrogate models have been generated for the output transient responses or for their envelopes and trend (as in the alternate approach), the regressive performance of the resulting neural networks needs to be assessed, in order to gain confidence on their approximating capabilities. The process is called the verification of the goodness of fit. The evaluation of the goodness of fit is done on the data that was used for training the neural networks (resulting from the DoE). This measures how well the training optimization process succeeded. In parallel, the goodness of fit is also evaluated for the random validation cases. This validation test measures how well the surrogate model can be generalized to the full design space.

The outcome of the evaluation of the goodness of fit is thus a margin of confidence with which the user may use the parametric surrogate models. If the goodness of fit is judged unsatisfactory, then the surrogate models should be rejected and the neural networks should be trained again, with different architectures. Checking the goodness of fit consists of verifying several fitting parameters and plots, as seen in Kirby [83].

The first statistic to check is the coefficient of determination (R-square, or R^2), which measures how much of the data is explained by the regression. By definition, R^2 is lower than 1 and can be negative for non-linear regression. An R^2 of 1 means that the regression fits the data perfectly, while a negative R means that the regression induces more error than just taking the mean of the data (cf. Equation 40):

$$R^2 \equiv 1 - \frac{SSE}{SST} \quad (40)$$

where SSE is the sum of squared errors (or residual sum of squares)

SST is the total sum of squares

SSE and SST are defined by the following equations:

$$SSE \equiv \sum_i \sum_j \left(Y_{i,j} - \hat{Y}_{i,j} \right)^2 \quad (41)$$

$$SST \equiv \sum_i \sum_j \left(Y_{i,j} - \bar{Y} \right)^2 \quad (42)$$

where $Y_{i,j}$ is the value of the response resulting from the test campaign

$\hat{Y}_{i,j}$ is the corresponding value predicted by the surrogate model

\bar{Y} is the mean of all the $Y_{i,j}$'s

Looking at the value of R^2 is not sufficient. Indeed, the fit can be perfect ($R^2=1$) while not representing the physics of the problem. For example, a sine wave can be

used to fit 3 points that are aligned, as illustrated in Figure 96, where both the green sine wave and the black line fit the red data points perfectly ($R^2=1$ in both cases). Thus, one of these regressions is misleading in terms of representation of the physics. This problem is particularly acute for the present study since the transient signals are often multiscale by nature. This phenomenon, where data points are regressed to the point that the physics of the problem is no longer captured, is called overfitting. In order to check how well the regression represents the physics of the problem, one must consider the random validation points and evaluate how well they are predicted by the regression equation. Therefore, R^2 is also evaluated while including the validation cases.

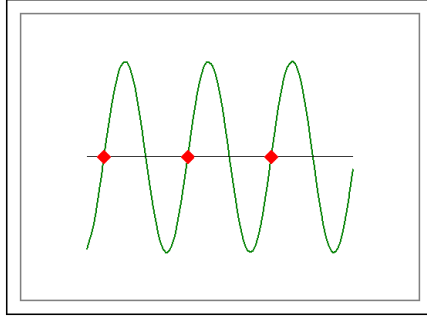


Figure 96: Illustration of Overfitting

The second way of measuring the goodness of fit is to look at the “Actual by Predicted” plot (cf. Figure 97). In this graph, each point represents a test case, training or validation. For each point, the x-axis value is the value \hat{y} predicted by the regression and the y-axis value is the actual one y , computed during the test campaign by the TDS model (and the envelope detection scheme for the alternate method). For a perfect regression, where those values are equal, the data point lies on the diagonal line $y = \hat{y}$. Therefore, for a good fit, the data points are distributed closely to the line. In red are the points of the training set, while the blue points correspond to the random validation test cases. Since the training points are used in

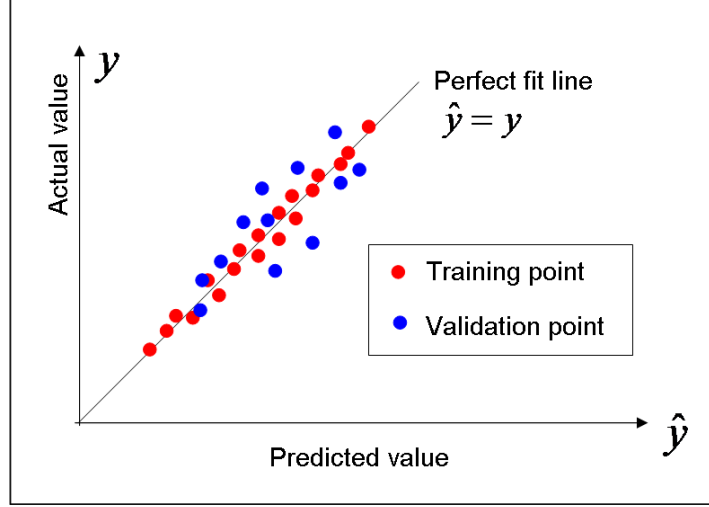


Figure 97: Actual vs. Predicted

the tuning and optimization of the neural networks, they are typically closer to the diagonal line than the validation points are.

The “Residual by Predicted” chart (cf. Figure 98) plots the residual error $e = \hat{y} - y$ (the difference between the regressed value and the actual value) against the predicted value of the response. For a good fit, the points should be randomly distributed around the vicinity of the horizontal line $e = 0$. If, on the contrary, an identifiable pattern emerges, then it can be inferred that the regression fails to take into account effects such as interactions among the variables.

Another way to look at the regression error is to plot the distribution of relative error, given in Equation :

$$Relative\ error = \frac{\hat{y} - y}{y} \quad (43)$$

Ideally, in order to have confidence in the model, the error distribution must have a normal distribution shape, centered on 0, and the standard deviation must be low. The desired shape for the error distribution is pictured in Figure 99. A low prediction error on the training points, indicated in the Model Fit Error (MFE), ensures that the regression is accurate. When validation points are included in the MFE, the Model

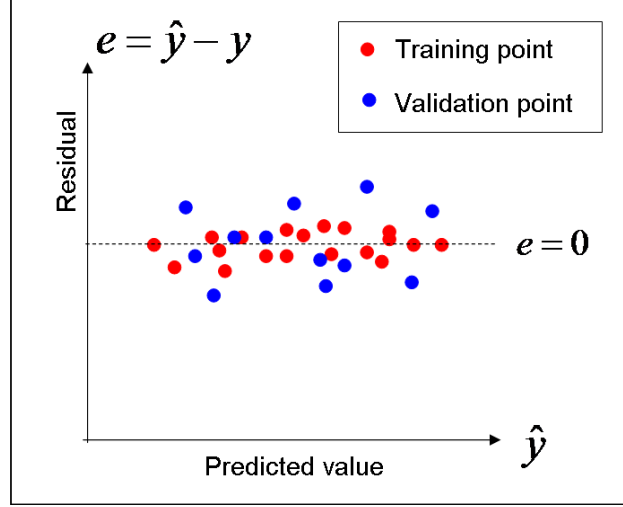


Figure 98: Residual vs. Predicted

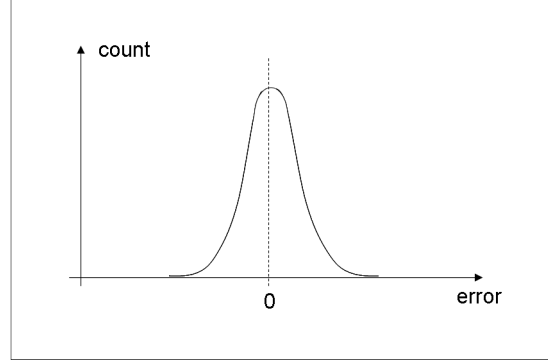


Figure 99: Desired Distribution Shape for MFE and MRE

Representation Error (MRE) is evaluated. A low prediction error on the validation points, indicated in the MRE, ensures that the regression can be interpolated and thus can be used as a good representation of the physics of the system. Thus, the MFE and MRE give a direct visualization of the confidence that the designer can have in the accuracy and representativity of the dynamic surrogate model.

5.2.6 Step 6: Perform a Monte-Carlo Simulation Using the Dynamic Surrogate Models

The outcome of Step 5 is a set of dynamic surrogate models that approximate the transient behavior of the dynamic system. The design process can now leverage the efficiency of those surrogate models to generate the behavior curves needed for the

thorough exploration of the design/operation space via the Time-Domain Filtered Monte-Carlo technique.

In Step 6, these behavior curves are generated with a Monte-Carlo simulation. As explained in Chapter II, uniform distributions are applied to the static input variables. The Monte-Carlo engine uses a random number generator to draw N_{MC} design/operation scenarios that sample the entire design/operation space. The number of random scenarios N_{MC} is chosen arbitrarily. It should be assigned a high enough value so that the generated design scenarios cover all regions of the design space.

For each of these scenarios, the dynamic surrogate models are run, taking advantage of their efficiency. Thus, the outcome of Step 6 is a set of N_{MC} static data points (one set per static response) as well as a set of time-domain signals. In the case of the wavenet-based methodology (from Hypothesis 3a), N_{MC} behavior curves per dynamic response are generated. In the alternate approach, based on the approximation of envelopes (from Hypothesis 3b), $3.N_{MC}$ behavior curves per dynamic response are generated: N_{MC} signals representing the trend of the transient response, N_{MC} signals representing the sup-envelope of the signal ripple, and N_{MC} signals representing the inf-envelope of the signal ripple.

5.2.7 Step 7: Populate the Interactive Visualization Environment

Once the behavior curves and the static data points have been generated by the Monte-Carlo simulation engine coupled with the dynamic surrogate models, they can now be imported into VisTRE, the interactive data farming visualization environment that was discussed in Chapter II (cf. Figure 42). In the case of the alternate approach stemming from Hypothesis 3b, the $3.N_{MC}$ behavior curves (per dynamic response) are combined in order to generate the N_{MC} sup-envelopes of the dynamic response and its N_{MC} inf-envelopes, in the process described in Figure 85. The N_{MC} reconstructions

performed in the Monte-Carlo Simulation is illustrated in Figure 100. The resulting visualization environment in VisTRE is illustrated in Figure 101. At this point, the stage is set for interactive design space exploration and the evaluation of dynamic transient constraints.

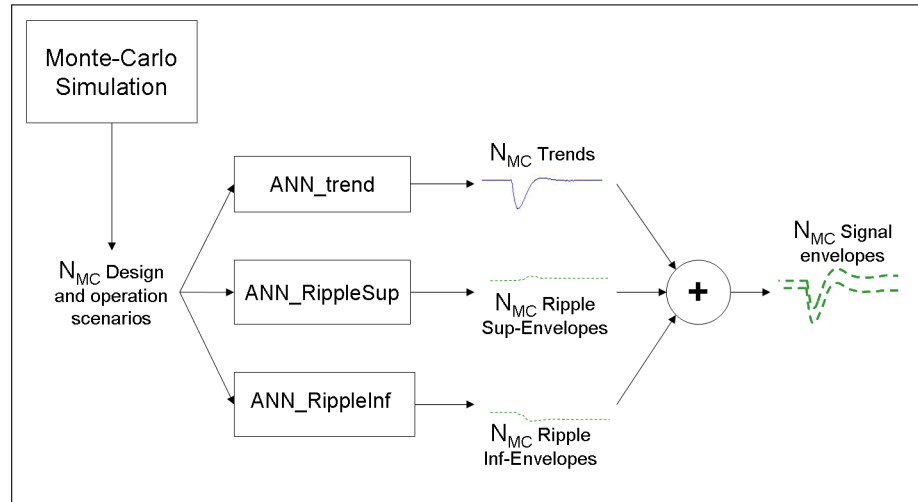


Figure 100: Envelope Reconstruction after the Monte-Carlo Simulation

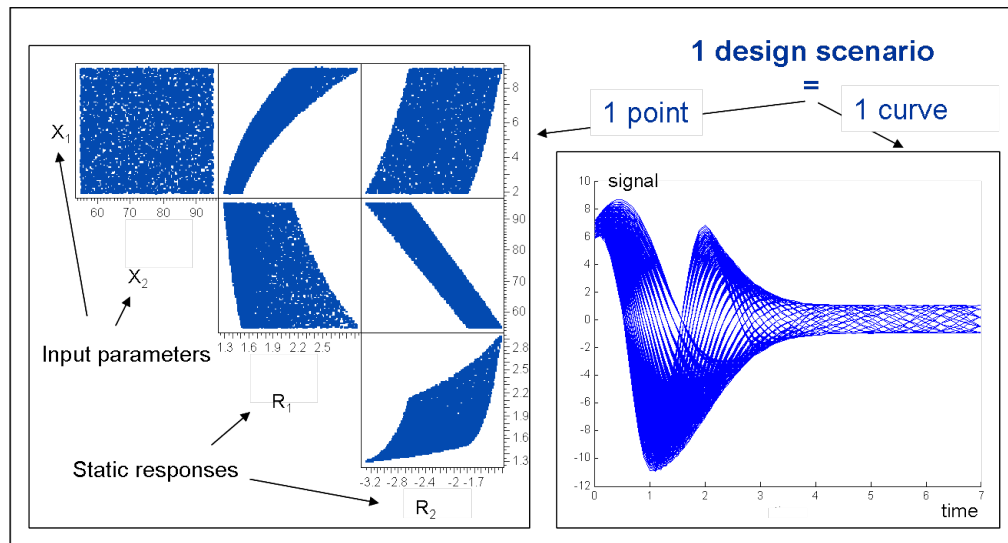


Figure 101: VisTRE Populated with the Outputs of the Monte-Carlo Simulation

5.2.8 Step 8: Explore and Filter the Design/Operation Space

At this stage, the behavior curves and the static data points have been imported into the interactive visualization environment VisTRE. The user can now take advantage of the capabilities of the environment to explore the design/operation space, and visualize correlations between parameters.

In addition to the insight on the design/operation space that it may offer, the interactive visualization environment allows the designer to add static and dynamic constraints, and filter-out the static design points and dynamic behavior curves that do not meet these constraints, as explained in Chapter II (cf. Figure 43). The result is therefore a set of design/operation points that meet the design and operation constraints. The designer can further restrict the design/operation space by selecting the design/operation points that correspond to optimal values of design criteria (e.g. minimal weight, minimal cost).

The value of the methodology is apparent: the reward for the effort put into generating dynamic surrogate models is the capability to instantly visualize design regions while ensuring that all constraints are met, whether they be static or dynamic. The interactive environment also enables the user to instantaneously vary the constraints and redo the design space exploration and filtering, thus providing some insight on the robustness of the optimal design region with respect to the uncertainties on the constraints.

5.2.9 Summary

The methodology presented in this chapter encapsulates the hypotheses formulated in this thesis proposal. Thus, the proposed methodology is in itself the expression of an implicit hypothesis:

Hypothesis 4 (H4): The proposed methodology fulfills the research objective, i.e. it integrates transient regime analysis and pertaining dynamic constraints

into the design specification of aircraft dynamic systems. The output of the methodology is an optimal design region that verifies the transient-related dynamic constraints.

The multi-step methodology outlined in this chapter is summarized in Figure 102.

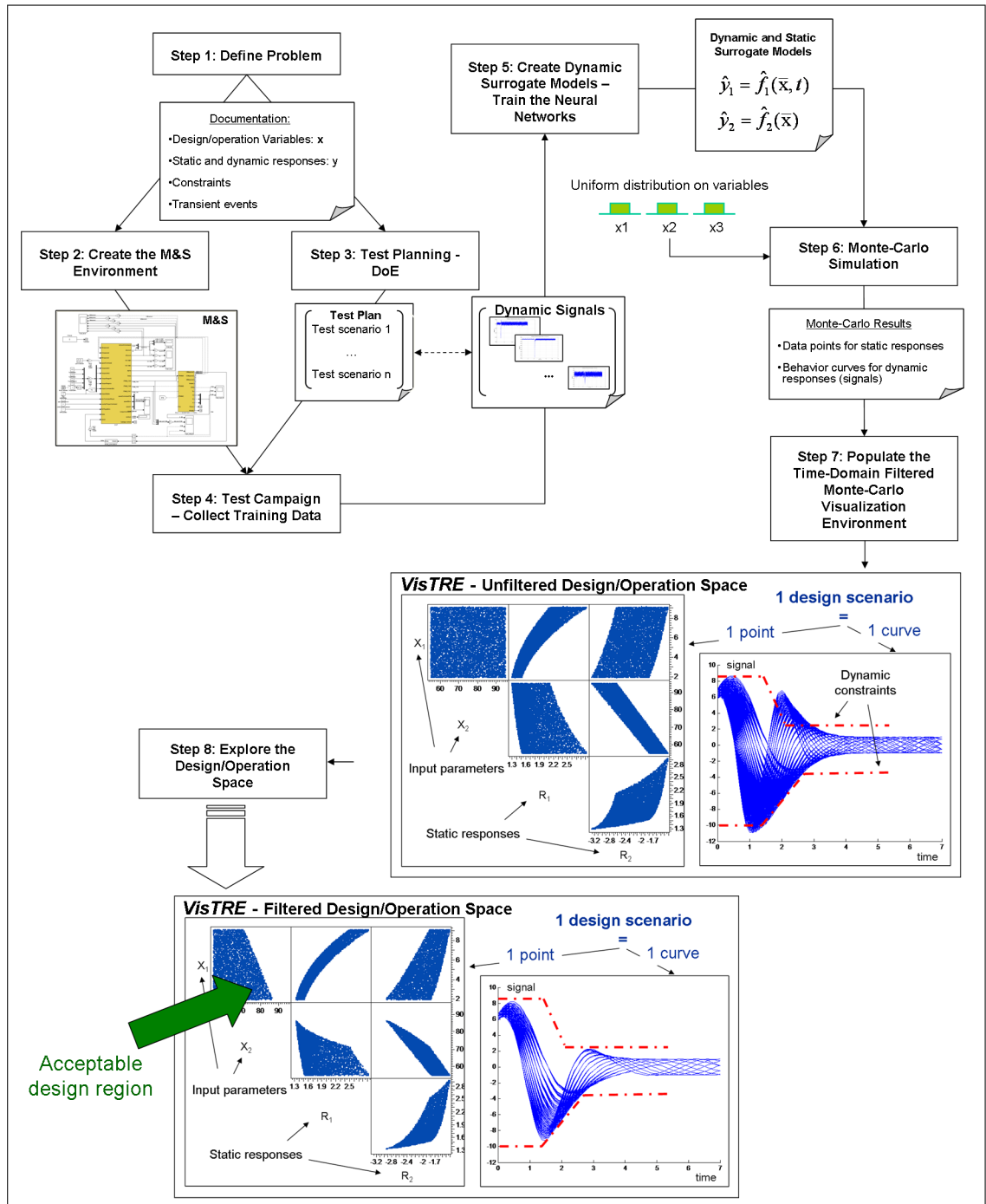


Figure 102: Methodology Overview

Chapter VI

METHODOLOGY IMPLEMENTATION AND PROOF OF CONCEPT

In the previous chapters, hypotheses were formulated as tentative answers to the research questions. Following the guidelines of the scientific process, these hypotheses, which have been translated into a methodology, need to be tested in order to infer their validity. Hypotheses are tested by experimentation. Thus, it is essential to properly *design the experiments* that will be performed in order to test the hypotheses.

Testing the hypotheses will be carried out in four progressively complex experiments, the later ones building on the successes of the previous ones.

6.1 Experiment 1: System Identification of a Simple Mathematical System

6.1.1 Objectives and Plan

The goal of this first experiment is to test hypothesis 3a regarding the system identification capabilities of wavenets. Recall hypothesis 3a:

Hypothesis 3a (H3a): A nonlinear system identification approach based on wavelet neural networks will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems.

Testing this hypothesis requires the development of a wavenet training module implementing the learning strategies laid out in the previous chapter. The wavenet training module will be implemented in Matlab [105]. For this first experiment, the

performance of the wavenet training module will be tested for a simple notional dynamic system experiencing a transient regime. The multivariate dynamic system will be *designed* so that its transient behavior is representative of a typical system, i.e. it exhibits oscillations that dampen with time.

Hence the mathematical expression of the simple dynamic system was chosen to be:

$$y = \left(x_1 \cdot e^{-\frac{(t-1)^2}{2}} + 1 \right) \cdot \cos(x_2 \cdot (t - 1)) \quad (44)$$

where $8 \leq x_1 \leq 10$ and $1 \leq x_2 \leq 3$

$0 \leq t \leq 10$

The function y is plotted with respect to time in Figure 103, for different values of x_1 and x_2 within their ranges of variations.

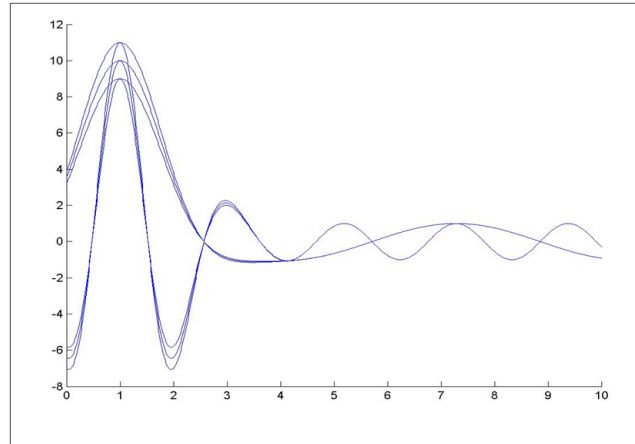


Figure 103: Representation of the Simple Dynamic System

In order to enable the generation of dynamic surrogate models, the wavenet should exhibit good approximation properties and should be efficient to run, once learning is over. The tasks to be performed for this experiments are listed below:

- Task 1.1: Build the wavenet training module

- Task 1.2: Train the wavenet for the system expressed in Equation 47 and verify the approximation performance of the wavenet training module.

Because the benchmarking of Chapter III concluded that wavenets might become more difficult to train as the number of variables increases, two test functions will be approximated by the wavenet.

First, x_2 will be fixed to the value 3. Thus, the function y becomes bi-dimensional, since it depends on t and on x_1 . Then, in a second stage, both x_1 and x_2 are allowed to vary within their range.

Thus, the complexity of the function to approximate increases incrementally, which allows for the study of how the complexity in the wavenet training increases.

- Task 1.3: Verify that the wavenet output is efficient to run

6.1.2 Implementation and Results

6.1.2.1 Task 1.1: Build the Wavenet Training Module

The wavenet training module was implemented in Matlab and is given in Appendix C. The optimization, by which the training module adjusts the wavenet coefficients in order to minimize the regression error, is carried out with the Matlab built-in function `fmincon`. The optimization algorithm used is active-set, based on sequential quadratic programming algorithms. The algorithms for active-set optimization are given in Gills et al. (1981) and Gills et al. (1991) [57, 58].

As suggested in Zhang and Benveniste [162], constraints are put on the optimizer in order to force the wavelets to have their support stay within limits: outside a predetermined range for time that encompasses the simulation time window, the wavelets should have values close to zero. Indeed, a wavelet has a compact support (or a nearly compact support): outside an interval $[a, b]$, its value is zero (or asymptotically close to so). If during training, this interval is allowed to drift too far from the time interval

of simulation, then the contribution of this wavelet to the output of the wavenet is null, and the wavelet would be carried by the optimizer like a “dead weight”.

6.1.2.2 Task 1.2: Train the Wavenet

As explained in the introduction of this section, the wavenet was trained for two systems:

- A reduction of the system described in Equation 47 to a bi-dimensional system, where time and x_1 vary and the variable x_2 is kept at a constant level $x_2=3$.
- The full three-dimensional system of Equation 47, where all three variables t , x_1 , x_2 vary.

Bi-Dimensional System

For the training cases, x_1 was varied between 8 and 10, by increments of 0.5. A total of 10 random validation cases was performed. Training gave satisfactory results for 10 wavelons. Table 10 lists, for each of the 10 wavelons, the resulting coefficients for the wavelon connection weight, the scaling and translation coefficients for the wavelet capturing the time t , and the scaling and translation coefficients for the wavelet capturing the design variable x_1 .

Table 10: 2D Wavenet - Table of Coefficients

Wavelon	Weight w_i	Wavelet t		Wavelet x_1	
		Scaling a	Translation b	Scaling a	Translation b
1	371.88	1.47	0.71	-0.10	37.29
2	88.95	1.19	0.55	0.35	-25.39
3	9.60	5.77	-1.04	-0.10	-10.33
4	4.55	8.83	-0.66	-0.10	-9.43
5	-325.90	1.59	-0.64	-0.08	-36.03
6	-115.77	-91.50	-18.94	-22.39	6.03
7	-25.32	5.74	-0.75	164.48	76.51
8	734.61	23.40	4.12	-2108.62	924.97
9	-156.42	117.17	158.35	-286.90	-704.52
10	75.45	-328.55	117.28	44.94	-23.09

As a first step of the evaluation of the goodness of fit, the result of the training process can be visualized in the actual vs. predicted plot shown in Figure 104, where the training cases are represented by red points and validation cases by blue points. One can see that the points fall on the perfect fit line almost exactly. This good fit is reflected in the R^2 values, which are extremely close to 1 (above 0.999) for both the training set and the validation set.

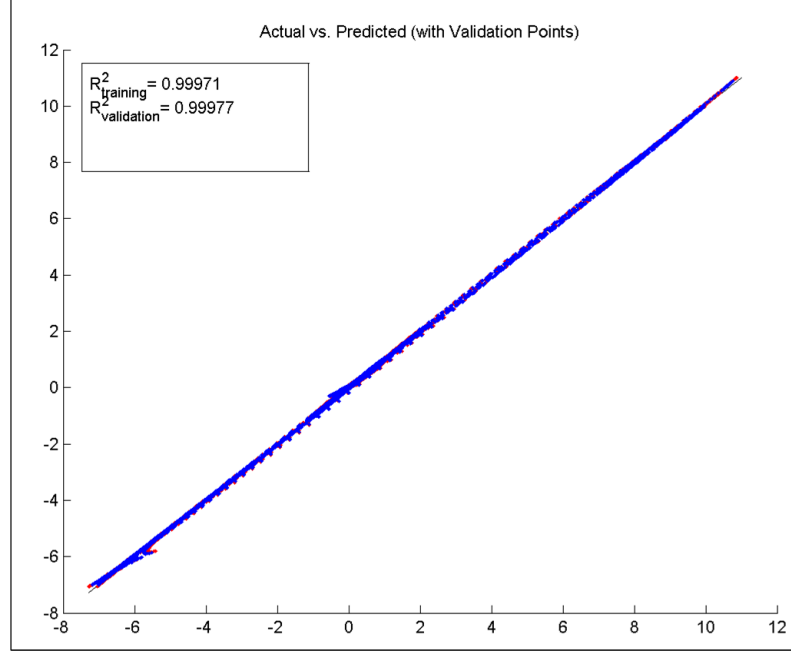


Figure 104: 2D Wavenet - Actual vs. Predicted

In the residual plot, shown in Figure 105, one can see that the residual oscillates in the vicinity of zero. However, some distinctive patterns appear, which is usually not desirable for static surrogate models. However, for dynamic surrogate models of time-domain signals, this does not necessarily indicate a bad fit, as explained below.

This presence of patterns in the Residual vs. Predicted plot for a dynamic surrogate model is natural since for each training (or validation) case i , the residual can actually be regarded as a signal $e_i(t)$, defined by Equation 45. Therefore, if for a training case, the signal varies continuously (in the sense that the output of the TDS model represents a continuous physical phenomenon), the residual points corresponding to a training case will belong to a continuous parametric curve defined by Equation 46, where Y_i and \hat{Y}_i are the actual signal of the i^{th} training case and its prediction respectively.

$$e_i(t) = \hat{Y}_i(t) - Y_i(t) \quad (45)$$

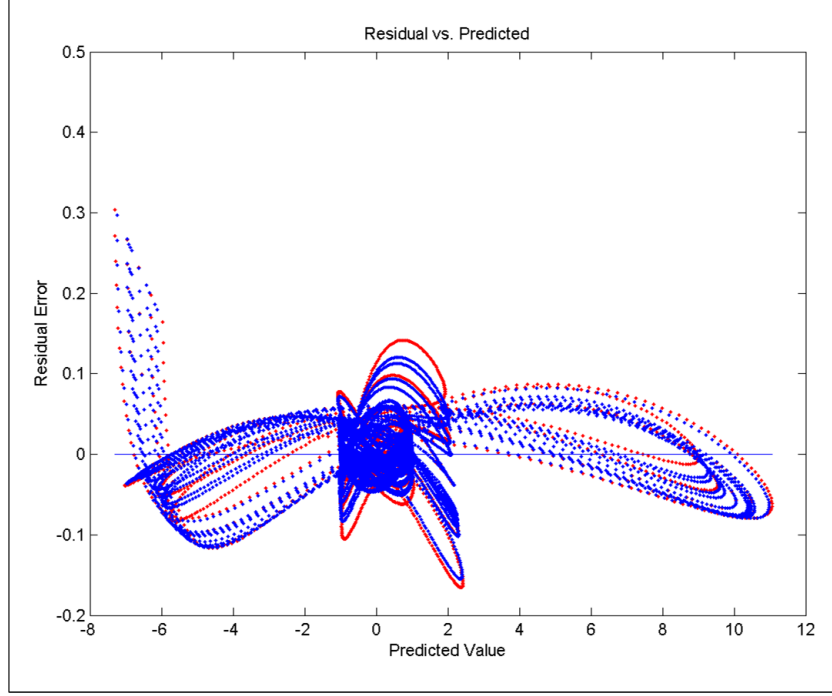


Figure 105: 2D Wavenet - Residual vs. Predicted

$$\begin{cases} x(t) &= \hat{Y}_i(t) \\ y(t) &= e_i(t) \end{cases} \quad (46)$$

In the residual vs. predicted plot of Figure 105, in addition to points that aligned to form parametric curves, one can notice the presence of a denser cluster of points for predicted values around zero. This is explained by the fact that the transient response oscillates around zero, as seen in Figure 103. As time increases, the oscillation amplitude stabilizes and reaches the steady-state regime. Thus, the set of points outside this cluster correspond to the transient regime.

Because the residual error can be regarded as a signal, it is useful to consider the residual vs. time plot, composed of an overlay of all residual signals. This allows for the visualization of the quality of the approximation with respect to time. If a particular time region presents systematically high values of residual, then the surrogate model fails to appropriately capture the behavior of the dynamic system for this time region. The residual vs. time plot for the 2D system is given in Figure

106. It is apparent that the residual signals oscillate around zero, which is desirable. However, the figure shows, that for the first instants after the transient perturbation, the wavenet underestimates the response value. In the present case, the residual remains within 0.3, which is low and therefore acceptable.

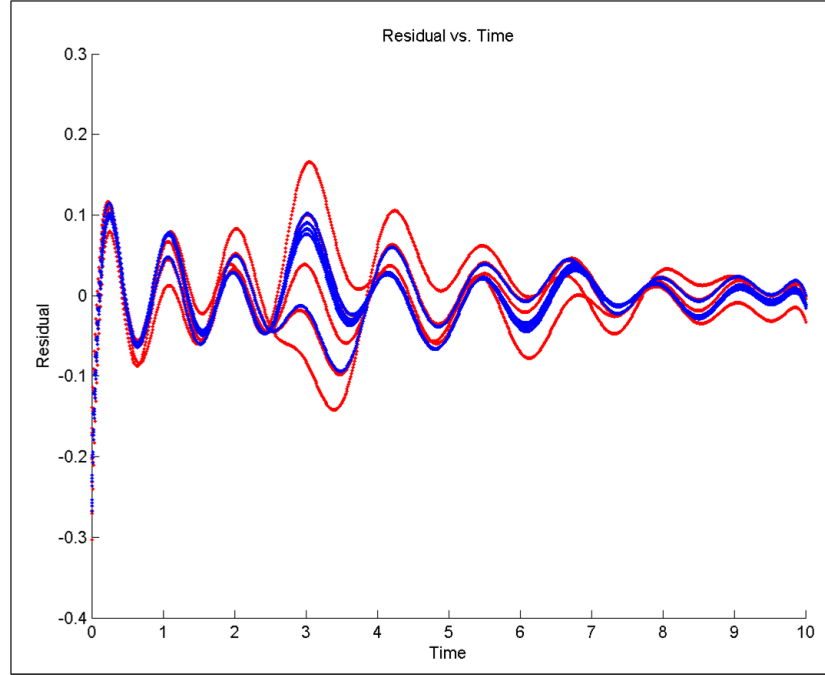


Figure 106: 2D Wavenet - Residual vs. Time

The distribution of the relative error is shown in Figure 107. As desired, the shape of the distribution is close to that of a normal distribution. However, the relative error induced by the regression reaches values higher than 4000%. This is due to the fact that the response signal takes values around zero. Therefore, the relative error around the points near zero can quickly reach high proportions. For instance, an prediction error of 0.01 when the actual value is 0.001 yields a relative error of 1000%. When considering that the values taken by the signal approximately range from -10 to 10, this error can be considered as low.

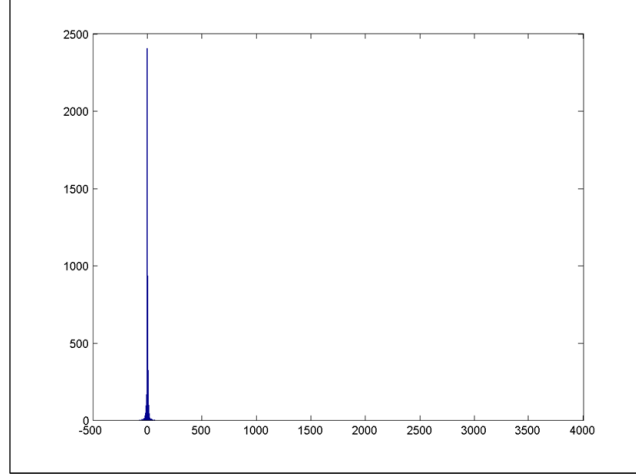


Figure 107: 2D Wavenet - MFE for the Relative Error

Therefore, for signals that take values close to zero, it is more appropriate to consider the MFE (distribution of errors on the training points) and MRE (distribution of errors on the training and validation points) for the absolute error (or residual) rather than the relative error. The plots of the corresponding MFE and MRE obtained after training of the wavenet on the 2D system are given in Figures 108 and 109.

One can see on the MFE and MRE figures that the error distributions have a shape similar to than of a normal distribution centered on zero, which is indicative of a good fit. The means and standard deviation are given in Table 11. Their values, close to zero, confirm the goodness of fit.

Table 11: 2D Wavenet - MFE and MRE Statistics		
	MFE	MRE
Mean	$\mu_{MFE} = -7.78 \cdot 10^{-5}$	$\mu_{MRE} = -1.61 \cdot 10^{-5}$
Standard Deviation	$\sigma_{MFE} = 6.24 \cdot 10^{-4}$	$\sigma_{MFE} = 3.38 \cdot 10^{-4}$

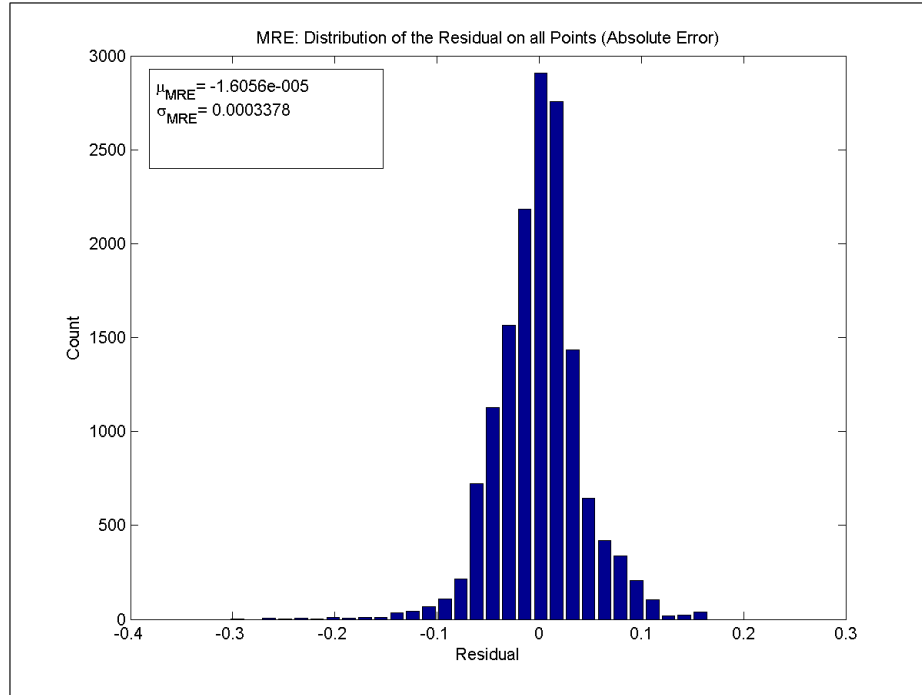


Figure 109: 2D Wavenet - MRE for the Residual Error

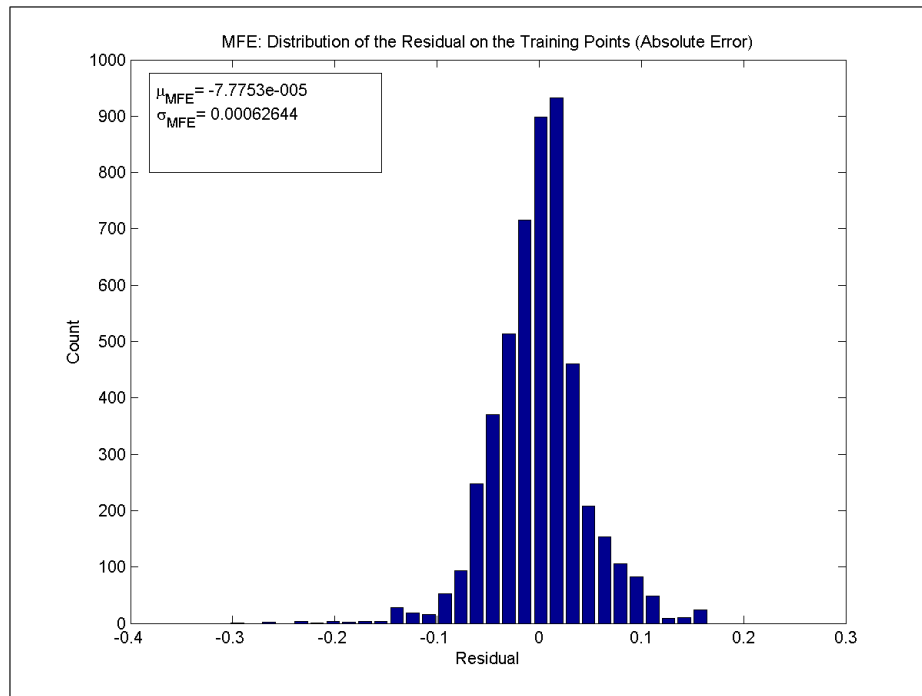


Figure 108: 2D Wavenet - MFE for the Residual Error

In a nutshell, the wavenet training with 10 wavelons on the 2D system showed satisfactory results. Indeed, the R^2 values are excellent, and the residual distributions exhibit the desired properties. Moreover, it was seen that the inherent peculiarities of dynamic responses make the study of the residual vs. predicted plot challenging. What would be indicative of poor fits for static responses are not necessarily bad for dynamic responses. Hence, it is necessary to also study the residual vs. time plot, which gives a better visualization of the regressive performance of the wavenet, and which highlights the time regions at which the wavenet regression yields questionable approximations. Thus, the set of tools to evaluate the goodness of fit for the surrogate modeling of dynamic responses has been expanded.

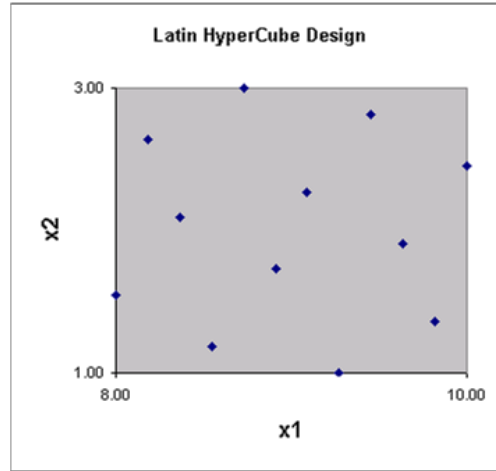
Tri-Dimensional System

In this experiment, one design variable has been added: the system now has three variables. The mathematical equation representing the system is given in Equation 47.

As described in step 3 of the methodology, a training dataset was generated with a Design of Experiments (DoE). For this purpose, a Latin HyperCube DoE with 12 runs was created in JMP, the statistical analysis and visualization software by SAS Institute [132]. The resulting DoE table is listed in Table 12 and visualized in Figure 110. One can see that the points generated by the LHC optimization process are well spaced across the design space. In addition to the 12 DoE runs for the training of the wavenet, 10 random validation cases were generated.

Table 12: DoE Table for the Tri-Dimensional Mathematical System

Case	x_1	x_2
1	8.18	2.64
2	8.00	1.55
3	8.55	1.18
4	9.64	1.91
5	9.45	2.82
6	8.36	2.09
7	9.82	1.36
8	9.09	2.27
9	8.91	1.73
10	10.00	2.45
11	8.73	3.00
12	9.27	1.00

**Figure 110:** LHC DoE Table for the Training of the 3D System

The addition of a dimension meant adding a term in the multidimensional tensor wavelet, for each wavelon. Moreover, the added complexity of the multivariate signal

to approximate necessitated a higher number of wavelons, in the same fashion as it requires more neurons for the training of traditional sigmoid-based neural networks. The combination of the higher number of wavelet terms and the higher number of wavelons results in a longer training time. Training the wavenet on the 3D system yielded the actual vs. predicted plot shown in Figure 111, for a total of 57 wavelons.

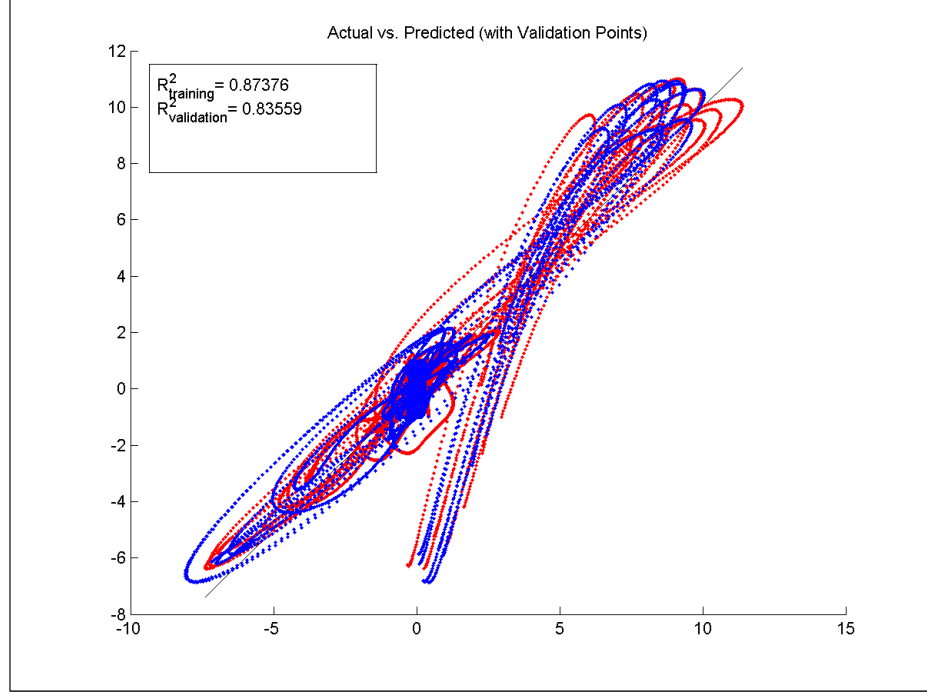


Figure 111: 3D Wavenet - Actual vs. Predicted

In the actual vs. predicted plot, one can see that the fit is not satisfactory, as there is a clear pattern of points that are overestimated (below the diagonal line) by the wavenet, despite R^2 values of 0.87376 and 0.83559 for the training and validation cases respectively, which is honorable. The patterns of points that are overestimated are visualized in the residual vs. time plot (cf. Figure 112), where one can see that it is the early time region (corresponding to the transient) that is overestimated (positive residual).

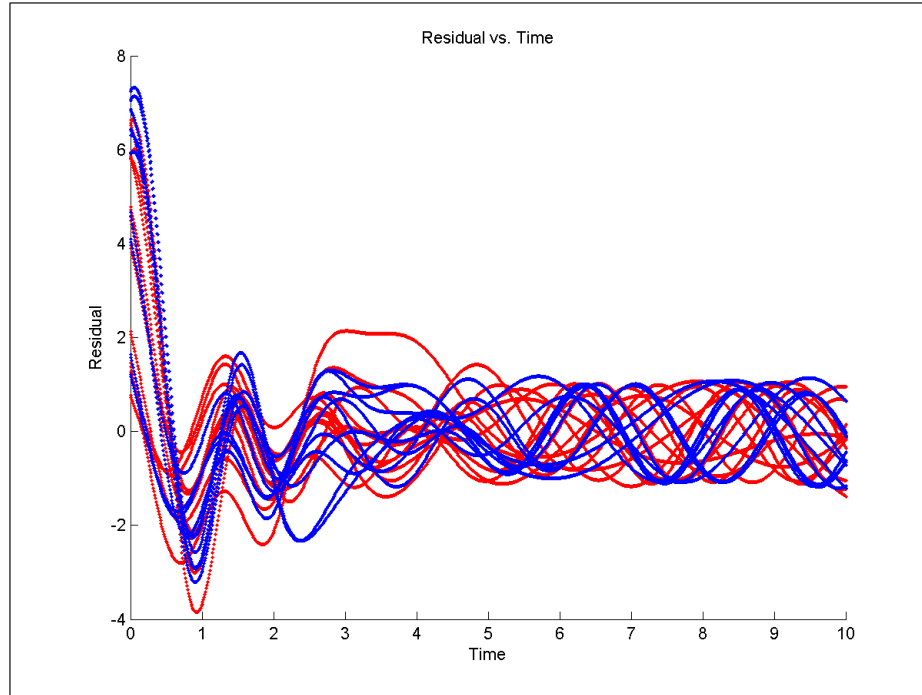


Figure 112: 3D Wavenet - Residual vs. Time

These results were obtained after a long training process, which lasted 3,955 seconds (nearly 66 minutes, compared to 4 minutes for the 2D case). The bad fit can be explained by the fact that the training procedure stalled into a local minimum, as depicted in Figure 113.

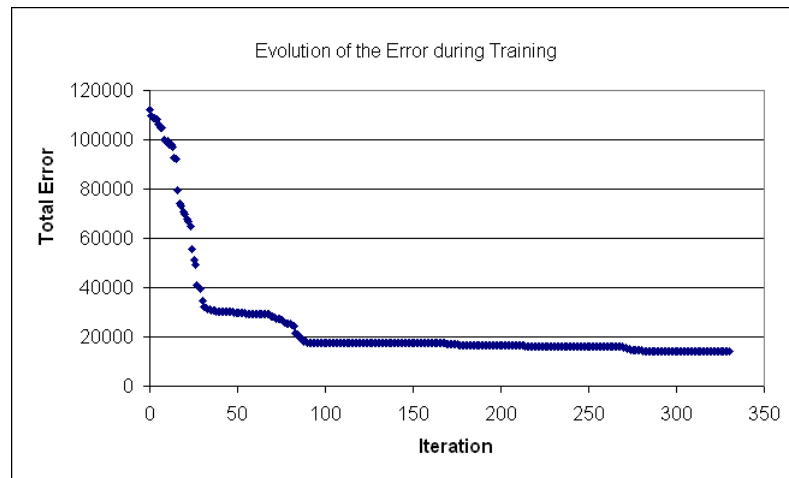


Figure 113: 3D Wavenet - Evolution of the Total Regression Error During Training

6.1.3 Summary

In this experiment, Hypothesis 3a was tested on a simple notional mathematical dynamic system. Indeed, the system identification approach based on wavelet neural networks (wavenets) was tested first on a reduction of the system to two variables (time and one design variables), and then on the full system with three variables (time and two design variables).

For the first test, the results showed close to perfect fit, thus confirming the validity of the mathematical formulation of wavenets. However, the addition of a design variable in the second test increased the complexity so much that the resulting fit was poor. The fact that the training algorithm fell into a local minimum suggests that a more efficient training optimization approach is needed for the implementation of wavenets.

The difficulties encountered with the training of wavenets leads to the conclusion that the envelope approach formulated in the alternate Hypothesis 3b should be preferred except for low numbers of dimensions.

6.2 *Experiment 2: Visual Transient Response Exploration*

6.2.1 Objectives and Plan

The goal of the second experiment is to test Hypothesis 2 and Hypothesis 1, which are recalled below:

Hypothesis 1 (H1): A data farming approach, based on the integration of time as a dimension in the the Filtered-Monte-Carlo approach, will conduce to the efficient and thorough exploration of the design/operation space for dynamic signals with dynamic constraints.

Hypothesis 2 (H2): Dynamic surrogate modeling of time domain simulation models will enable the efficient implementation of the Time-Domain Filtered Monte-Carlo technique for the exploration of design/operation space of dynamic systems subject to dynamic constraints.

The testing of Hypotheses 1 and 2 is performed on the simple 2D mathematical example discussed above. The 2D equation representing the notional system is recalled in Equation 47.

$$y = \left(x_1 \cdot e^{-\frac{(t-1)^2}{2}} + 1 \right) \cdot \cos(3 \cdot (t - 1)) \quad (47)$$

In addition to the dynamic response $y(t, x_1)$, a static response R will be added to the notional dynamic system, as defined in Equation 48, where x_2 is an additional design variable:

$$R(x_1, x_2) = \frac{(x_1 + x_2)^2}{x_1} \quad (48)$$

The input of Experiment 2 is the dynamic surrogate models generated by the wavenets of Experiment 1, and defined by the wavenet coefficients listed in Table 10. Testing Hypothesis 2 amounts to verifying that the Monte-Carlo simulation process, designated as “Step 6” in the methodology, takes little time to complete using the dynamic surrogate models.

Testing Hypothesis 1 includes testing the applicability and usefulness of the Visual Transient Response Explorer (VisTRE), the proposed interactive visualization environment for the rapid filtering of transient signals. Thus, the interactive visualization environment VisTRE first needs to be developed. This will be attempted in JMP, the statistical software package by the SAS Institute [132], in order to leverage the capabilities of the software for static parameters. It will have to be shown that with the visualization environment, the design/operation space is entirely sampled

(thorough exploration) and that once the visualization environment is populated, the interaction with the user is quick enough to enable instant “what-if” analyses on the dynamic signals and on the constraints.

The tasks to be performed for Experiment 2 are:

- Task 2.1: Run the Monte-Carlo simulation and verify the speed of the process
- Task 2.2: Develop the interactive visualization environment for Time-Domain Monte-Carlo Filtering
- Task 2.3: Design Space Filtering. The aim is to verify that dynamic constraints can be rapidly added or changed, and that behavior curves are easily filtered

6.2.2 Implementation and Results

6.2.2.1 Task 2.1: Run the Monte-Carlo Simulation and Verify the Speed of The Process

Since only one design variable is considered, the Monte-Carlo Simulation, instead of using random cases, was performed by discretizing the range of variation ($x_1 \in [8, 10]$) uniformly into 200 cases. Then, for each of these cases, the wavenet was run in order to generate the corresponding 200 signals.

For these 200 Monte-Carlo cases, the total duration of the simulation using the wavenet surrogate models took a total of 0.175 seconds. This is of the same order of the time needed to run the 200 cases for the original equations, since these were chosen in order to be simple. However, compared to the simulation run times for TDS models, which can reach dozens of minutes, this is satisfactory.

6.2.2.2 Task 2.2: Develop the Interactive Visualization Environment (VisTRE) for Time-Domain Monte-Carlo Filtering

The Visual Transient Response Explorer (VisTRE) was implemented in JMP, using the native scripting language JSL, thus taking advantage of the readily available built-in function for multivariate scatterplot matrices. Thus, the task consisted of creating

The elementary constructs of JMP are data tables, such as the example given in Figure 114, where two design variables x_1 and x_2 are given for 3 design cases (the rows of the table). A third column represents the response $y = x_1 + x_2$.



Another main feature of JMP is the possibility to define scripts in the native scripting language JSL. These scripts allow to automate the manipulation of data tables and the creation of corresponding visualization plots.

184

each dynamic signal Y^k (e.g. voltage, current, or instantaneous temperature), the outputs of the Monte-Carlo simulation are aggregated into a single JMP table, called Monte-Carlo Signal Table (MCST), as illustrated in Table 13.

Table 13: JMP Monte-Carlo Signal Table (MCST) for Response Y^k

t	MC Case 1	.	.	.	MC Case m
t₁	$Y^k_{1,1}$				$Y^k_{m,1}$
t₂	$Y^k_{1,2}$				$Y^k_{m,2}$
.	.				.
.	.				.
.	.				.
t_n	$Y^k_{1,n}$				$Y^k_{m,n}$

Then, a main JMP table, referred to as the Monte-Carlo Main Table (MCMT), is created. The MCMT groups all static outputs from the Monte-Carlo simulation. Thus, they include, for each Monte-Carlo case, the values of the static design (and operation) variables, as well as the values of the static responses. In addition to that, the MCMT includes one column per dynamic response Y^k , containing the path of the corresponding Monte-Carlo Signal Table.

Table 14: JMP Monte-Carlo Main Table (MCMT)

MC Case	x₁	x₂	R₁	R₂	Y₁(t)	Y₂(t)
1	C:\JMPFiles\Y1.JMP	C:\JMPFiles\Y2.JMP
2		
3		
:	:	:	:	:		

Once the output data from the Monte-Carlo simulation is imported in JMP and segregated into Monte-Carlo Signal Tables and a Monte-Carlo Main Table, the visualization environment VisTRE, described in Chapter II, can be generated. For this purpose, a script, called “Create VisTRE” was created in JSL. The script contains the procedure to create VisTRE, as well as procedures that control the interactive transient analysis features that are implemented in VisTRE.

When called, the script “Create VisTRE” reads the MCMT in order to get the paths of each MCST. Then, for each Y^k , the script collapses all the data of the MCST into a single table, denoted $Y^k_Aggregate$, depicted in Table 15.

Table 15: JMP $Y^k_Aggregate$ Table

MC Case	t	Y^k
1	$t_{1,1}$	$Y^k_{1,1}$
	:	:
	$t_{1,n1}$	$Y^k_{1,n1}$
:	:	:
i	$t_{i,1}$	$Y^k_{i,1}$
	:	:
	$t_{i,n1}$	$Y^k_{i,ni}$
:	:	:
m	$t_{1,1}$	$Y^k_{m,1}$
	:	:
	$t_{1,nm}$	$Y^k_{m,nm}$

Then, for each $Y^k_Aggregate$ table, the built-in function Overlay is called, which creates an overlay plot for the aggregate table. This is an interactive feature, so that the user can select points in the overlay plots, which highlights the corresponding rows of the data table. JMP also enables to change the colors of selected points, or hide them, which will be useful for the subsequent filtering of data. The correspondence

between the $Y^k_Aggregate$ table, for the dynamic response Y^k , and its overlay plot, for the visualization of the transient behavior, is illustrated in Figure 115.

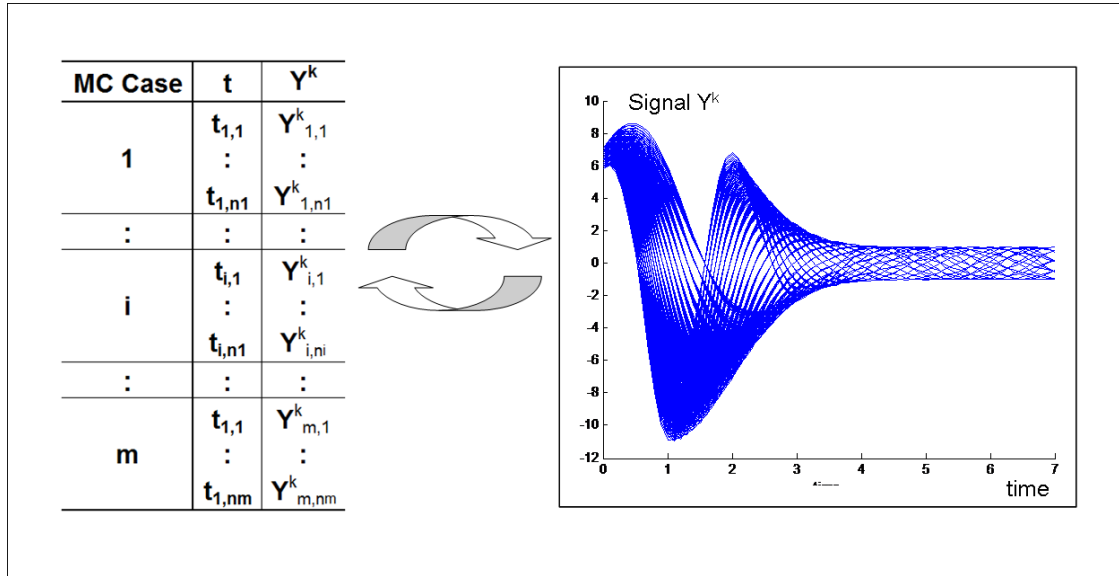


Figure 115: Interaction between the $Y^k_Aggregate$ Table and its Overlay Plot

The next step is to generate the scatterplot matrix for the static responses. The script “Create_VisTRE” reads the Monte-Carlo Main Table (MCMT), and for the variables and responses that are identified as static, it calls the built-in function “Scatterplot”. This generates the desired multivariate scatterplot, which is dynamically linked to the MCMT, as illustrated in Figure 116.

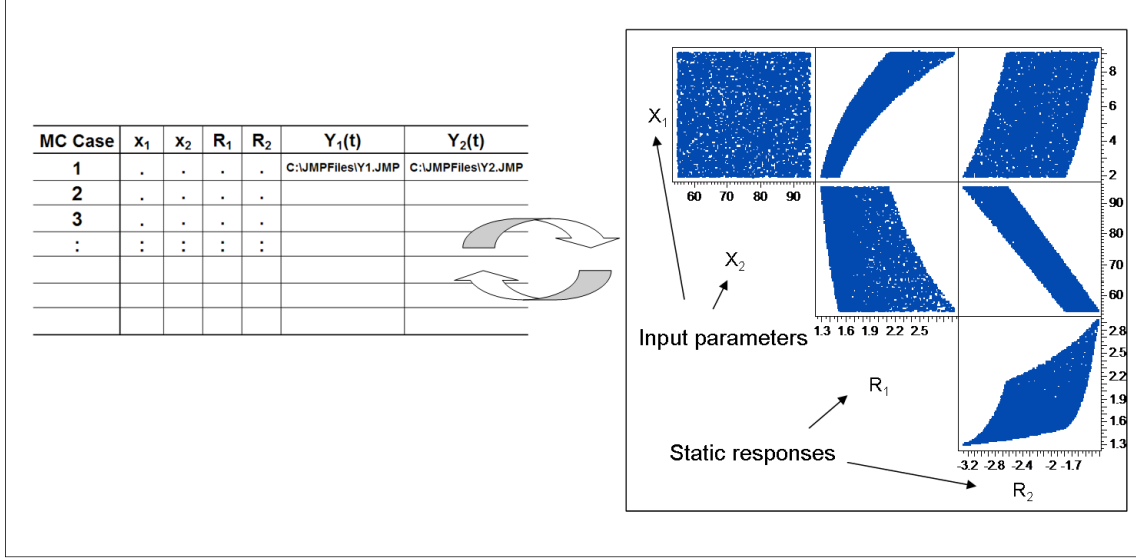


Figure 116: Interaction between the MCMT and the Scatterplot

At this stage, the main scatterplot for the static parameters as well as all the overlay plots for the visualization of the transient signals are generated. Each of these plots are linked to a data table. The next step performed by the script “Create_VisTRE” consists in implementing the links between plots, so that changes in one chart can be propagated to the other charts and tables. For this purpose, a set of action buttons is created by “Create_VisTRE” in each visualization chart, as depicted in the snapshots of the ViSTRE environment in Figures 117 and 118 (where the dynamic signal is the voltage). One can see that the scatterplot represented here is a symmetrical square matrix instead of a triangular one, so that for any combination of parameters p_1 and p_2 , both plots p_1 vs. p_2 and p_2 vs. p_1 can be visualized as cells in the scatterplot.

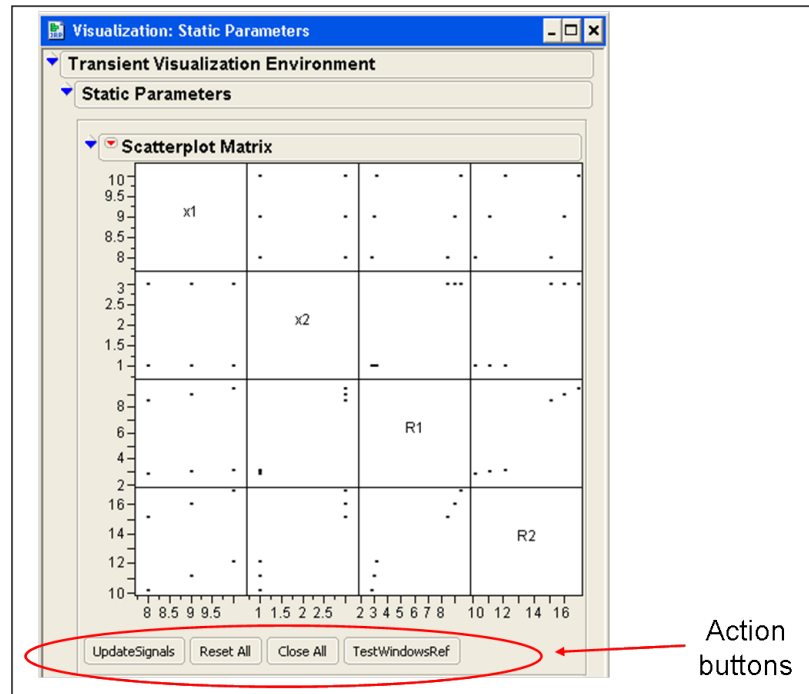


Figure 117: VisTRE: Scatterplot for Static Parameters

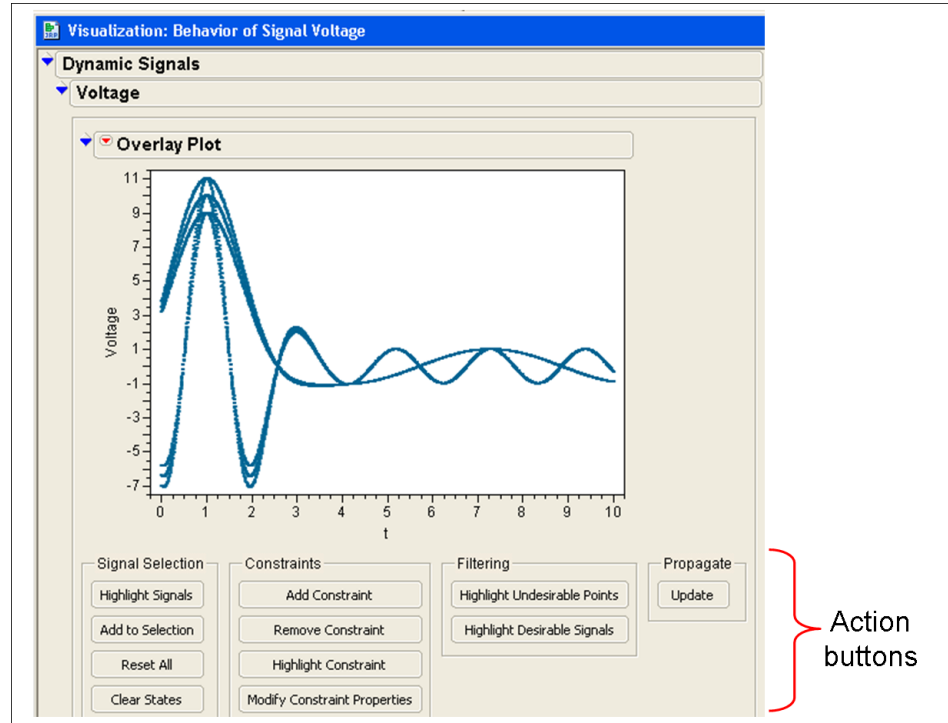


Figure 118: VisTRE: Overlay for Dynamic Transient Response (Voltage)

When performing design space exploration and filtering with VisTRE, the designer thus interacts with the plots (scatterplot or overlay) first, and then clicks on the action buttons in order to propagate the changes to the other windows. Thus, clicking an action button calls a specific subscript of “Create_VisTRE”, developed for a particular task. The most important button is the “Update” button, which calls a subscript that propagates the state of the plot to all the other plots (overlays or scatterplot).

One of the elementary tasks is the visualization of the full signals corresponding to specific points of interest in the overlay. For this purpose, a button called “Highlight Signals” is implemented for the time-domain overlays. When calling this button, the corresponding subscript of Create_VisTRE reads the points highlighted in the overlay plot, reads the corresponding Monte-Carlo case numbers in the associated Aggregation Table (cf. Figure 115), and searches for all the rows of the aggregation table that correspond to the case numbers. Then these rows are selected by the scripts, thus selecting all the time instants for the Monte-Carlo case numbers. Because the overlays are automatically linked with the aggregation table, the corresponding signals are highlighted. If the designer presses the button “Update”, the signals corresponding to the selected Monte-Carlo cases are highlighted in the other time-domain overlays, and the points corresponding to the selected Monte-Carlo cases are highlighted in the scatterplot. This is illustrated in Figure 119, where the selected signals and points appear in black. Conversely, the user can select design points in the static scatterplot, and update all charts (by clicking the action button “Update Signals” so that the corresponding signals in the time-domain overlays are highlighted.

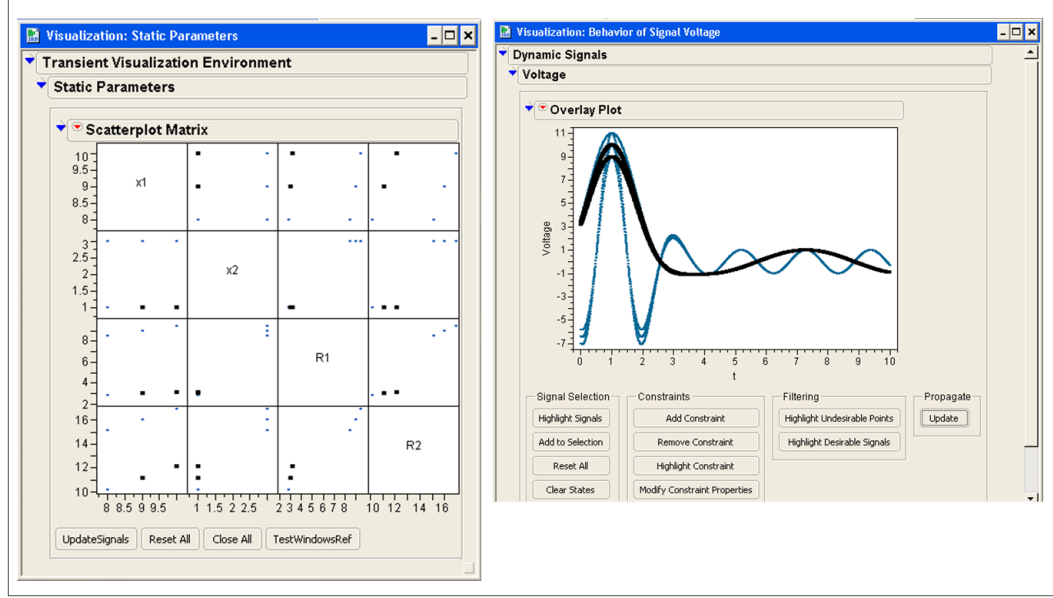


Figure 119: VisTRE: Highlighting Signals and Design Points

In order to implement the proposed Time-Domain Filtered-Monte-Carlo, the designer should have the possibility to quickly and easily add constraints, whether they be static constraints in the scatterplot, or dynamic constraints in the time-domain overlays. For this purpose, a set of action buttons are created in each time-domain overlay plot. These buttons call scripts that prompt the designer to either manually enter the points defining the vertices of the dynamic constraints, or load a predefined dynamic constraint, defined in a JMP table with two columns (time and constraint value). After adding a constraint, it is superimposed as a new red signal in the overlay plot, as seen in Figure 120.

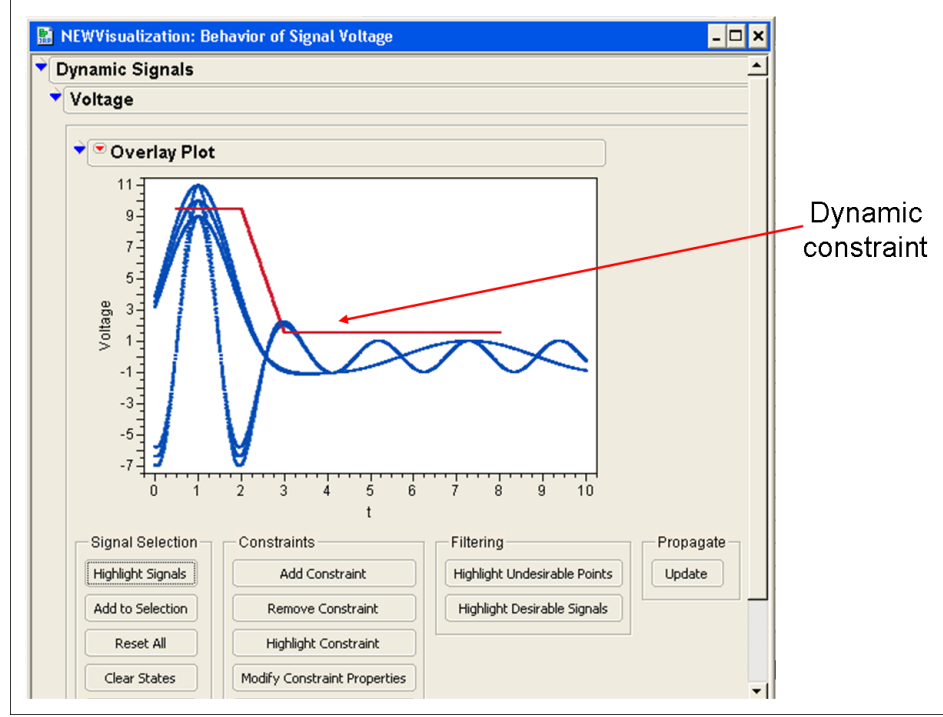


Figure 120: VisTRE: Adding a Dynamic Constraint for a Dynamic Response

By pressing the appropriate buttons, the user can modify the dynamic constraints, assign them roles (minimum or maximum constraint), and activate or deactivate them. The scripts control the constraints for each dynamic response Y^k by means of an additional data table $Y^k_Constraints$, which lists all the constraints defined for Y^k with their attributes. Furthermore, for each constraint, a column is created in the $Y^k_Aggregate$ table, listing the value of the constraint for each row (which corresponds to a time instant for a particular Monte-Carlo case). This is illustrated in Table 16. The addition of this column will be useful when finding the points that violate the dynamic constraints and filtering the corresponding signals.

Table 16: JMP Y^k _Aggregate Table after Adding Constraint C

MC Case	t	Y^k	C
1	$t_{1,1}$	$Y^k_{1,1}$	$C_{1,2}$
	$:$	$:$	$:$
	$t_{1,n1}$	$Y^k_{1,n1}$	$C_{1,n2}$
$:$	$:$	$:$	$:$
i	$t_{i,1}$	$Y^k_{i,1}$	$C_{i,2}$
	$:$	$:$	$:$
	$t_{i,n1}$	$Y^k_{i,ni}$	$C_{i,ni}$
$:$	$:$	$:$	$:$
m	$t_{1,1}$	$Y^k_{m,1}$	$C_{m,2}$
	$:$	$:$	$:$
	$t_{1,nm}$	$Y^k_{m,nm}$	$C_{m,nm}$

After adding the constraints and defining their attributes, the designer can find the points in the time-domain overlays that violate the constraints. This is done by pressing the action buttons developed for this purpose. These buttons call scripts that compare the dynamic response and the constraints for each row of the aggregate table (Table 16). Then the points that are found to violate the constraints are highlighted, as depicted in Figure 121.

Once the points violating the constraints are highlighted, the entire signals corresponding to these points can be highlighted (using the “Highlight_Signals” button) and hidden. Then, the designer can update all charts to propagate the change. Thus, only the signals meeting the dynamic constraints and the corresponding design points in the scatterplot are kept, as illustrated in Figure 122.

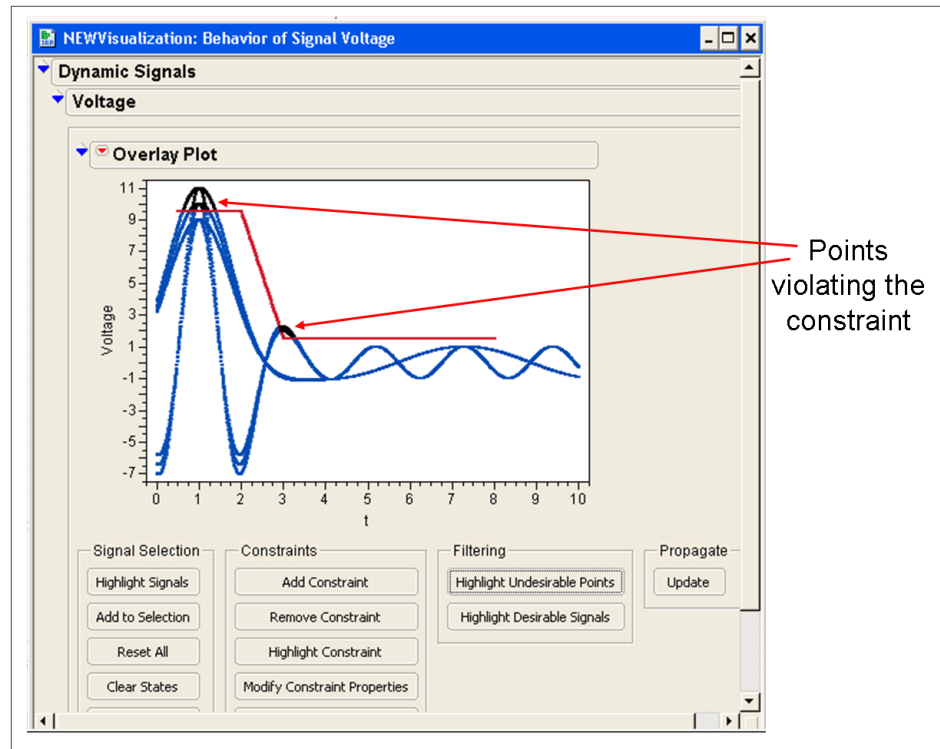


Figure 121: VisTRE: Finding the Points Violating the Dynamic Constraint

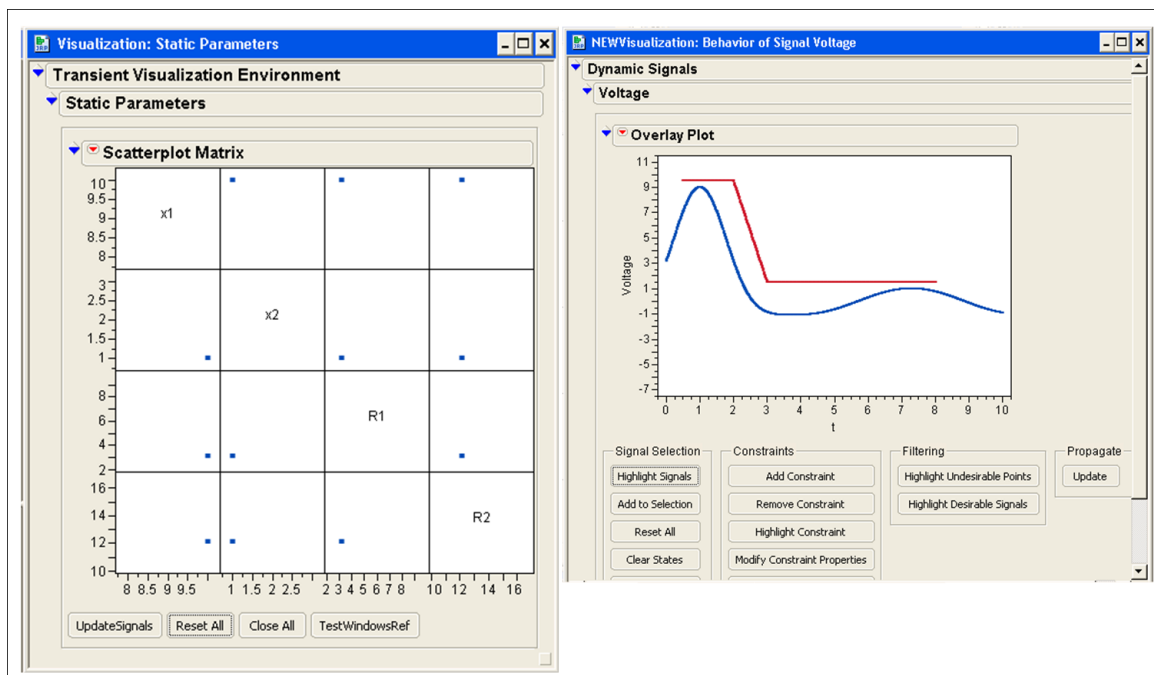


Figure 122: VisTRE: Filtering the Signals the Meet the Dynamic Constraint

In a nutshell, a script “Create_VisTRE” was created in JSL, the native scripting language of JMP, to generate the interactive visualization plots for the scatterplot and time-domain overlay plots, as well as action buttons that enable the user to interact with the plots and propagate the changes. These buttons call subscripts of Create_VisTRE that manipulate the background data tables that control the visualization plots. By interacting with the plots and performing a series of simple tasks with the action buttons, the designer can filter the design space so as to keep (or hide) only the signals (and their corresponding design points in the scatterplot) that meet the dynamic constraints. The script “Create_VisTRE” is given in Appendix D.

6.2.2.3 Task 2.3: Design Space Filtering

The procedure for design space filtering under dynamic transient constraint was applied to the system defined by Equations 47 and 48. The signals that violated the constraints were able to filtered out in the fashion shown in Figure 122.

6.3 Experiment 3: Proof of Concept - Implementation of the Methodology for the Design of an Electrical Network

In this experiment, the full methodology is implemented for the design of a simple 350 VDC electrical network, in order to test Hypothesis 4, which is recalled below:

Hypothesis 4 (H4): The proposed methodology fulfills the research objective, i.e. it integrates transient regime analysis and pertaining dynamic constraints into the design specification of aircraft dynamic systems. The output of the methodology is an optimal design region that verifies the transient-related dynamic constraints.

6.3.1 Objectives and Plan

The electrical network that will serve as a proof of concept for the methodology is a simple 350 VDC electrical networks, composed of a controlled generator that feeds power to an actuator motor. The variations of mechanical torque demanded by the actuator to the motor induce variations of power demand from the motor, which in turn induce transient perturbations on the electrical network. Thus, power quality constraints will apply to the network voltage, which has to be controlled in order to be maintained at a level of 350VDC. The network is depicted in the Simulink block diagram of Figure 123.

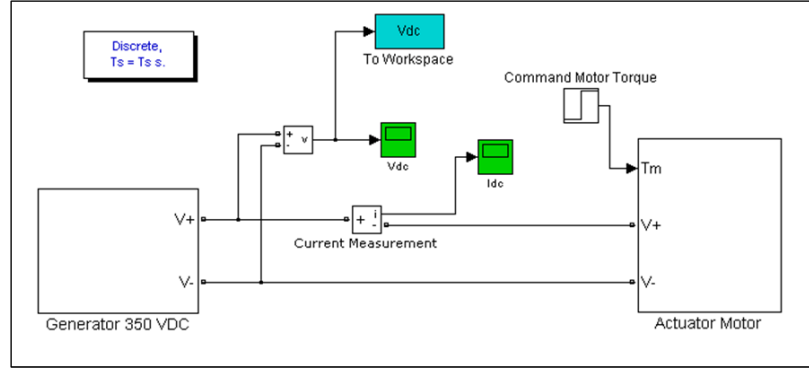


Figure 123: Simulink Diagram of the 350VDC Network

The candidate architecture for the generator system is comprised of several sub-systems, as depicted in Figure 124. First, a Permanent Magnet Generator (PMG), which converts mechanical power provided by a rotating shaft into a three-phase alternating electrical signal. The three-phase current is then converted into a DC voltage by an Insulated-Gate Bipolar Transistor (IGBT) rectifier. IGBT's are fast-switching power devices that are increasingly used in power electronics [82].

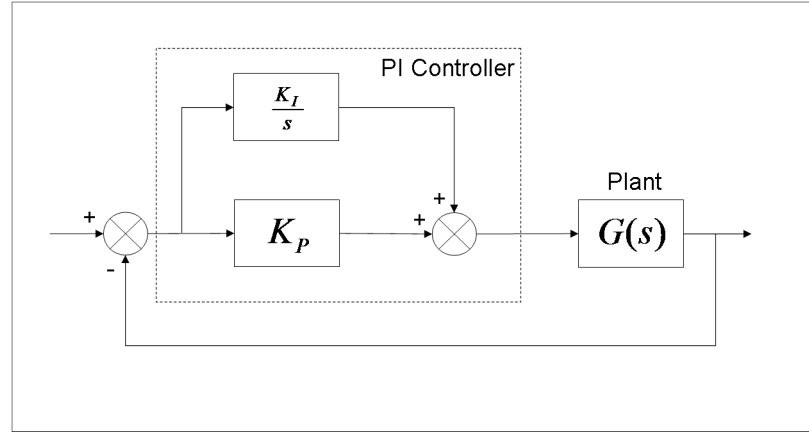


Figure 125: Block Diagram of a PI Controller

The actuator motor system is composed of a Permanent Magnet Motor (PMM), fed by a three-phase electrical signal that is the output of an inverter, which converts DC current into alternating three-phase currents. The inverter implemented here is based on Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFET), which are used to switch or amplify signals. The block diagram of the Simulink model representing the actuator motor system is depicted in Figure 126.

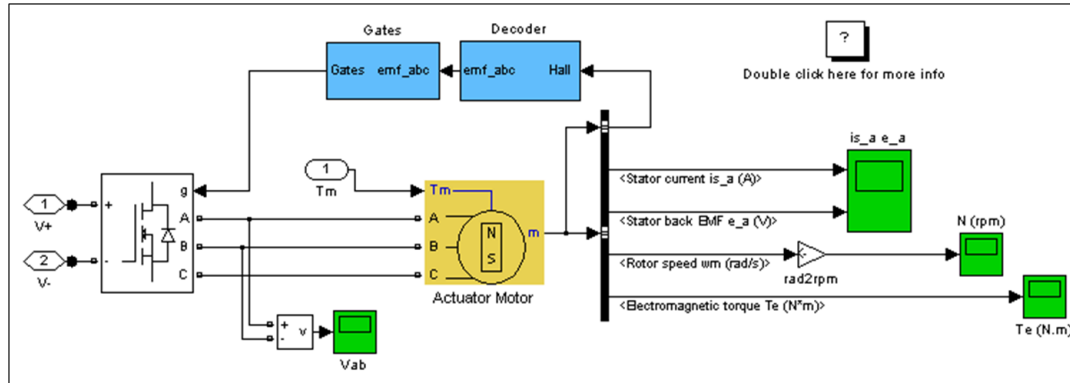


Figure 126: Simulink Diagram of the Actuator Motor

For the purpose of the experiment, the network voltage will be subject to the transient dynamic constraint pertaining to power quality given in Figure 5 and recalled in Figure 127.

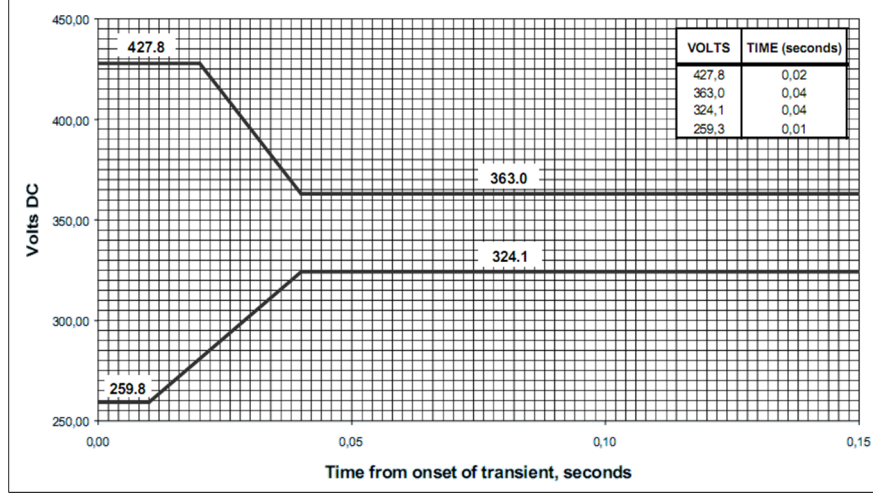


Figure 127: Power Quality Transient Constraint for the 350VDC Network Voltage

6.3.2 Implementation and Results

The methodology formulated in Chapter V of this thesis was applied to the design of the 350VDC electrical network described above. The implementation is described below for each step.

6.3.2.1 Step 1: Define the Problem

In this step, the problem is defined: the goal of the design activity is stated, the responses (static and dynamic) are identified, as well as the design and operation variables that might have an impact on the responses. Constraints and requirements are identified as well.

The goal of the design of the 350VDC electrical network is to specify the design characteristics of the generator and its controller, as well as those of the motor, in order to minimize the electrical losses of the generator. The electrical loss (“*loss*”) is represented by a simple model, taking the average value of the instantaneous Joules losses induced by the phase currents i_A , i_B and i_C in the stator of the Permanent Magnet Generator.

The design variables belong either to the generator subsystem or to the motor subsystem. Their names have the prefix “G” or “M”. Within these two groups, the design variables can further be classified into controller parameters (prefix “Gcontr”), rectifier parameters (“GRect”), permanent magnet generator (“GPM”), inverter parameters (“MInv”), and permanent magnet motor (“MPM”). This illustrates once more that the controller parameters are treated like design characteristics of the whole generator system. A total of 17 design variables were identified. They are listed in Table 17, along with a short description and their ranges of variation.

Table 17: Design Variables for the 350VDC Electrical Network

Variable name	System	Description	Min	Max
GContr_K	Generator controller	Proportional gain	-2.00E-01	-1.00E-02
GContr_I	Generator controller	Integral gain	-5.00E+00	-5.00E-01
GRect_Rs	Generator rectifier	Snubber resistance (Ω)	5.00E-01	5.00E+00
GRect-Cs	Generator rectifier	Snubber capacitance (F)	5.00E-06	1.20E-05
GRect_Ron	Generator rectifier	Internal resistance (Ω)	1.00E+01	2.00E+02
GPM_Rs	Generator PM machine	Stator phase resistance (Ω)	2.00E-01	5.00E+00
GPM_Ls	Generator PM machine	Stator phase inductance (H)	1.00E-03	1.00E-02
GPM_Flux	Generator PM machine	Flux linkage established by magnets (V.s)	1.50E-01	5.00E-01
GPM_J	Generator PM machine	Inertia (kg.m^2)	1.00E-04	2.00E-03
MInv_Rs	Motor inverter	Snubber resistance (Ω)	1.00E+04	5.00E+04
MInv-Cs	Motor inverter	Snubber capacitance (F)	1.00E-05	5.00E-05
MInv_Ron	Motor inverter	Internal resistance (Ω)	1.00E-03	1.00E-02
MPM_Rs	Motor PM machine	Stator phase resistance (Ω)	1.00E-01	5.00E+00
MPM_Ls	Motor PM machine	Stator phase inductance (H)	1.00E-03	1.50E-02
MPM_Flux	Motor PM machine	Flux linkage established by magnets (V.s)	5.00E-01	1.20E+00
MPM_J	Motor PM machine	Inertia (kg.m^2)	1.00E-04	2.00E-03
M_torque	Motor	Mechanical torque command (N.m)	2.00E+00	5.00E+00

As explained in the previous section, the network voltage is subject to a dynamic transient constraint. It will thus be a dynamic response on which dynamic surrogate models will be generated. Since the number of design variables is high, the alternate approach is used here: the design space exploration will be carried out in order to ensure that the envelope of the network voltage remains within the dynamic transient constraints.

As explained in Chapter I, when the power consumption of an electrical load decreases suddenly, the network voltage will experience an overvoltage. The shape and the value of the overvoltage depends on the control laws of the voltage source as well as on the characteristics of the load, its level of power consumption before the drop, and the level of the latter. Thus, in this experiments, the transient perturbation induced by the motor is modeled by a step increase in the mechanical torque demand from the actuator to the motor.

6.3.2.2 Step 2: Create the Modeling and Simulation Environment

A Modeling and Simulation (M&S) environment was created in Simulink. The corresponding block diagrams are given in the previous paragraphs (Figures 123, 124, and 126).

A duration of 2 seconds will be simulated for each run of the TDS model. The transient perturbation (step increase in motor torque) occurs at the instant $t = 1\text{s}$. The behaviors of the generator rotor speed and of the network voltage are plotted in Figures 128 and 129. At instant $t = 0$, all systems are off. The generator starts: the permanent magnet generator rotor speed increases, which produces a rise in voltage. The controller then forces the generator to adjust its rotor speed in order to stabilize itself at 350VDC. At $t = 1\text{s}$, the actuator motor experiences a step load in its mechanical torque. This induces an increase in power demanded from the generator, and a network voltage drop. In order to maintain the level of voltage to 350VDC, the controller then increases the generator rotor speed.

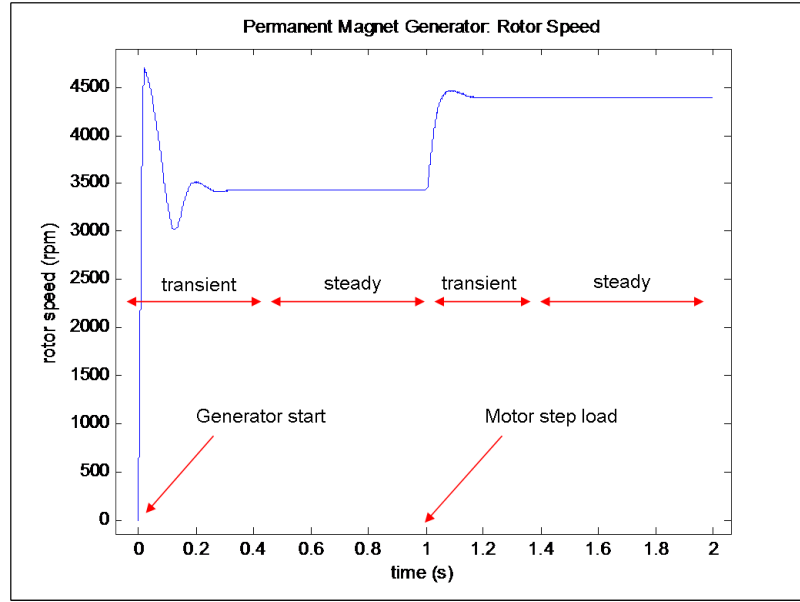


Figure 128: Generator Rotor Speed Behavior

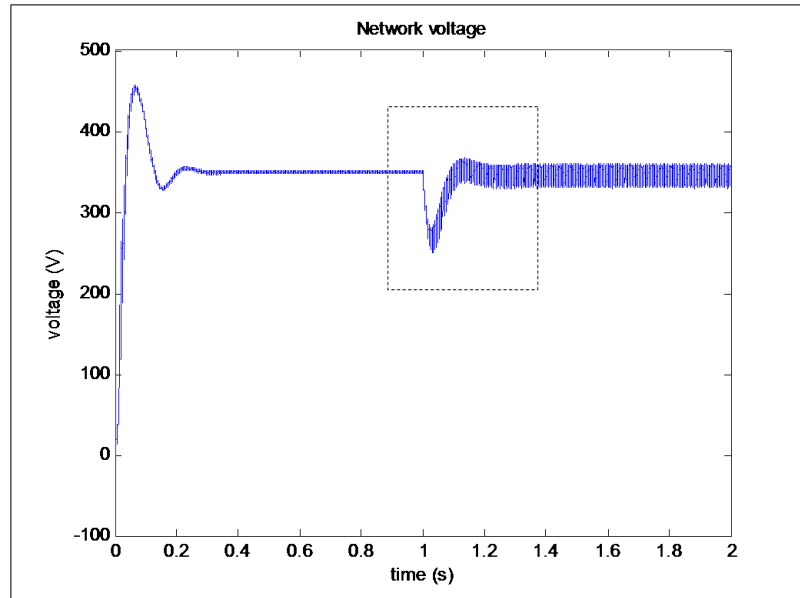


Figure 129: Network Voltage Behavior

Thus, the network voltage experiences two transient regimes: a first one, corresponding to the time when the generator starts, and a second one, induced by the step load increase in the motor torque. It is the latter, delimited by the dashed lines in

Figure 129, which will be studied in this experiment, and on which the power quality transient constraint will be enforced. The constraint is depicted on a zoomed plot of the network voltage in Figure 130.

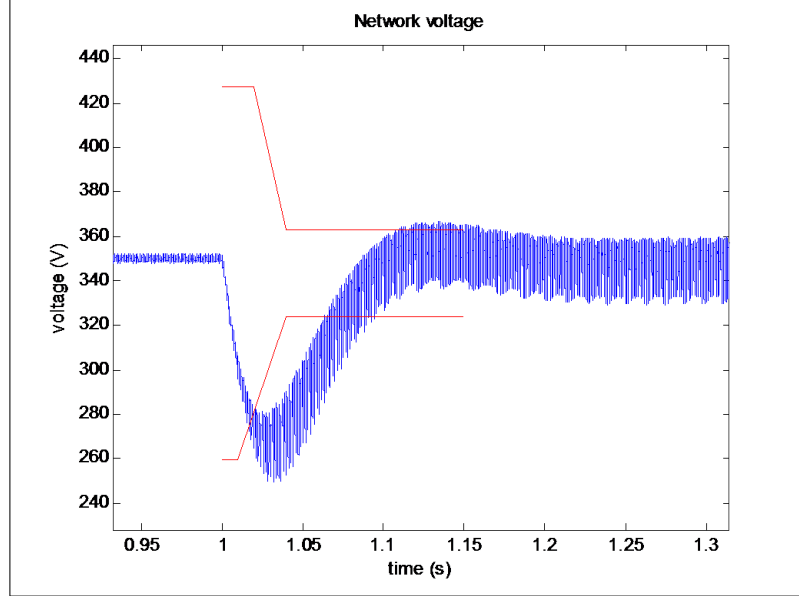


Figure 130: Network Voltage: Focus on the Transient Region

The simulation, performed in Simulink, is of the discrete-event type. A next-event time advance mechanism is used, where the simulation time step varies in an adaptive manner. The numerical integration solver for the ODE's is the Dormand-Prince method, which is a member of the Runge-Kutta class discussed in Chapter II.

6.3.2.3 Step 3: Test Planning - Design of Experiments

In this step, the test table is derived using a Design of Experiments (DoE). A 17-variable Latin HyperCube design was generated and optimized for 200 runs, using the JMP software. The optimization process took 17 minutes.

In addition to the test cases generated by the DoE, which will be used to train the neural networks, 50 validation cases were generated randomly. Therefore, the test table, which is the output of Step 4, is composed of 250 cases.

6.3.2.4 Step 4: Test Campaign - Collect the Training Data

Step 4 consists of running the cases defined in the test table resulting from step 3, and post-processing the results of the simulation runs in order to prepare the data for the training of the neural networks in Step 5.

The alternate method resulting from Hypothesis 3b is applied. Therefore, as explained in Chapter V, after each run, the sup-envelope and the inf-envelope are computed using the envelope extraction scheme described in Chapters IV and V. A multiresolution analysis (MRA) is performed on the voltage response (truncated to the time region of interest), in order to separate the signal into an approximation (trend) and a detail (ripple) signal. After a few preliminary runs, it was determined that the MRA yielded a denoised approximation after 10 decomposition levels. The resulting trend signal and ripple signal for a test case are plotted in Figure 131. The sliding windowing method is applied to the detail signal in order to get its inf-envelope and its sup-envelope.

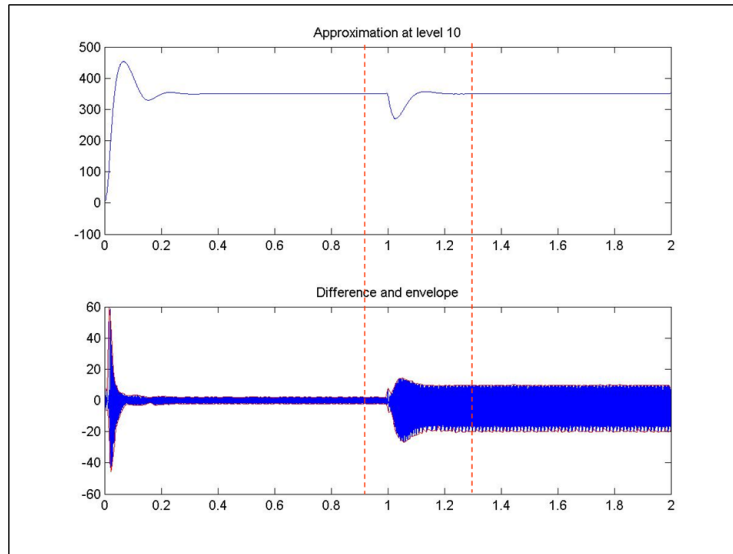


Figure 131: MRA at Level 10 on the Voltage Response

The outputs of step 4 are the data tables listing the values of the static responses and the behaviors of the voltage trend (V_{trend}), the voltage ripple sup-envelope

(VripSupEnv), and the voltage ripple inf-envelope (VripInfEnv). For these dynamic responses, the DoE table was rearranged in order to include the different time instants at which each signal was sampled, in the fashion depicted in Figure 91.

The simulation run times and the post-processing times are shown in Figure 132. With the notable exception of the first case, which took longer due to the initialization of variables, the simulation run times revolved around 34 seconds per case, while the post-processing times fell in the vicinity of 0.75 seconds per case. The total time for the test campaign and the post-processing amounted to 8,729 seconds, i.e. 2 hours and 25 minutes.

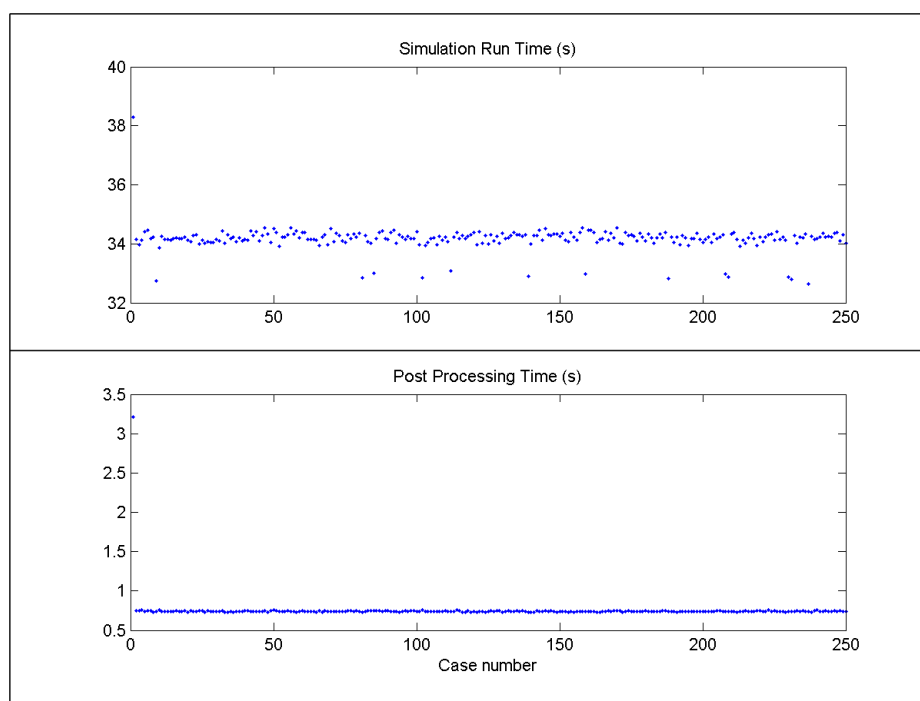


Figure 132: Test Campaign: Simulation Run Times and Post-processing Times

6.3.2.5 Step 5: Create Dynamic Surrogate Models - Train the Neural Networks

In step 5, the neural networks for the static response (Loss) and for the three signals (Vtrend, VripSupEnv, and VripInfEnv) are trained. Training was performed using a neural network training suite, developed at the Aerospace Systems Design Laboratory (ASDL) at the Georgia Institute of Technology, called the Basic Regression Analysis

for Integrated Neural Networks (BRAINN) [74]. BRAINN is a Matlab suite that integrates most functionalities implemented in the Neural Network toolbox of Matlab. It enables the designer to quickly select values for the number of neurons in the hidden layer, particular optimization algorithms for the training process, and other optimization parameters such as convergence threshold values. A screenshot of the BRAINN user interface is given in Figure 133.

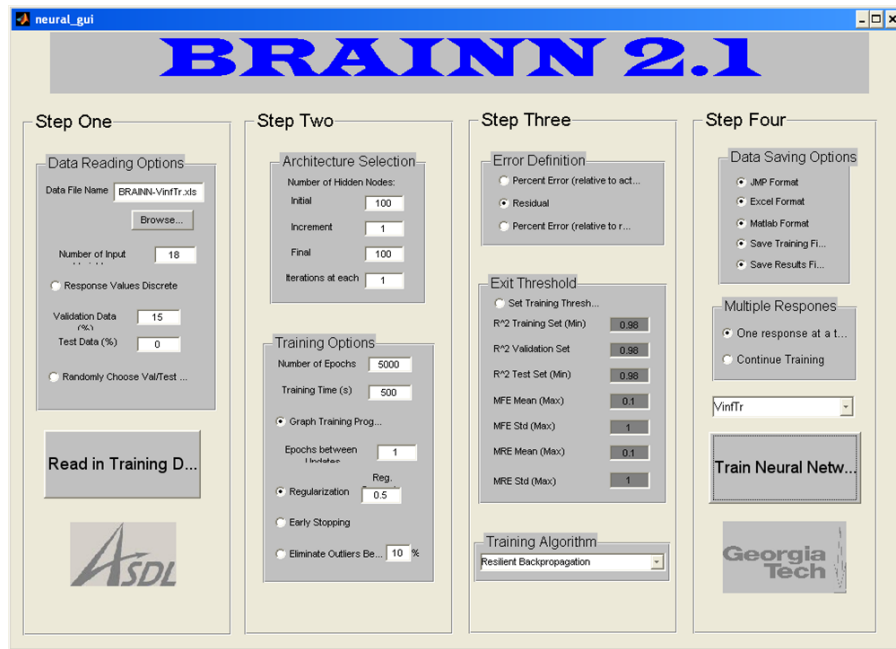


Figure 133: BRAINN User Interface [74]

The process for training the neural networks is that described in Chapter V: for each response, the neural network is trained for different values for the number of neural nodes in the hidden layer. The neural network architecture exhibiting the best fit is retained.

The static response Loss presented values that spanned a large interval. In order to facilitate the training, a transformation was applied to the response so that its logarithm was regressed. The results of the training process are then validated by performing the evaluation of the goodness of fit. The results for each response are presented below.

Static Response: Loss

The results of the fit for the natural logarithm of the electrical loss ($\ln(loss)$) is given in Figure 134, obtained for a neural network architecture with 4 neurons in the hidden layer. The Rsquare values are close to 1, the points fall closely to the perfect fit line in the Actual vs. Predicted plot, and the distributions of the relative error have the desired shape, with low means and standard deviations, for both the MFE and the MRE.

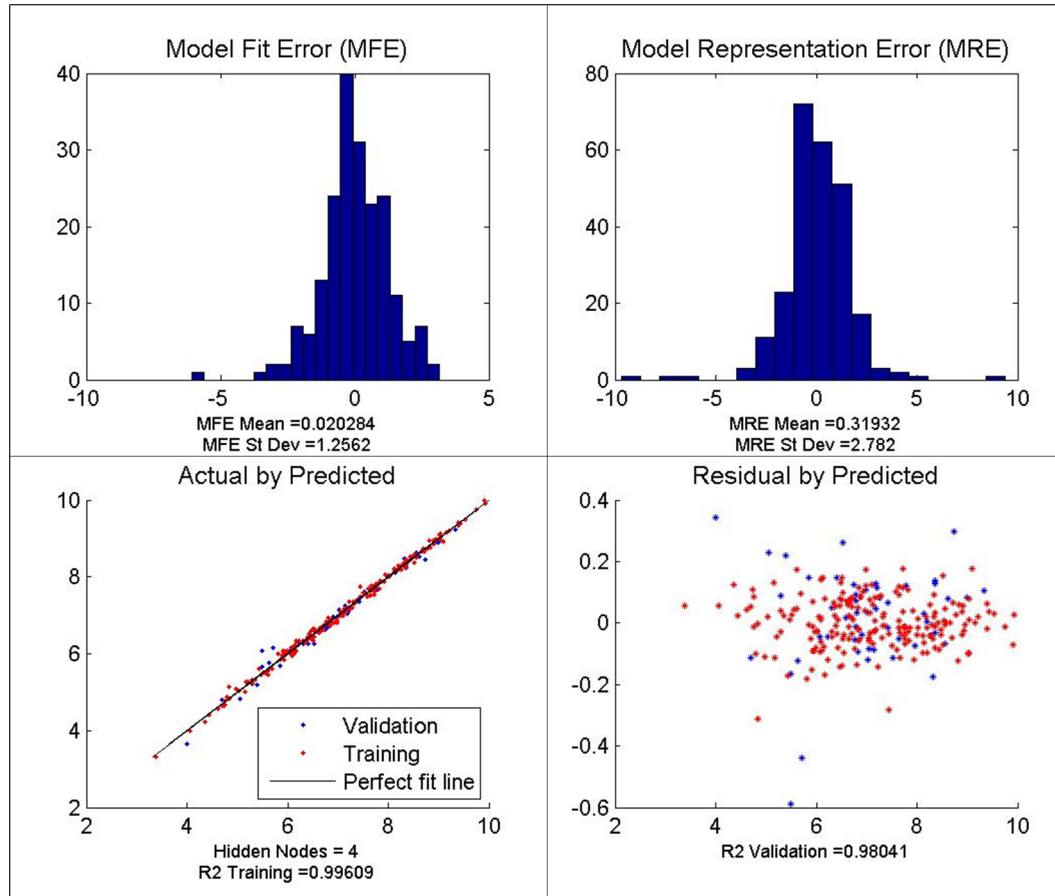


Figure 134: Goodness of Fit for $\ln(loss)$

Dynamic Response: Vtrend

Vtrend is the approximation of the network voltage, obtained after a wavelet-based MRA at level 10. The results of the training process are visualized in the

charts of Figure 135. The optimal neural network architecture comprises 15 neurons in the hidden layer.

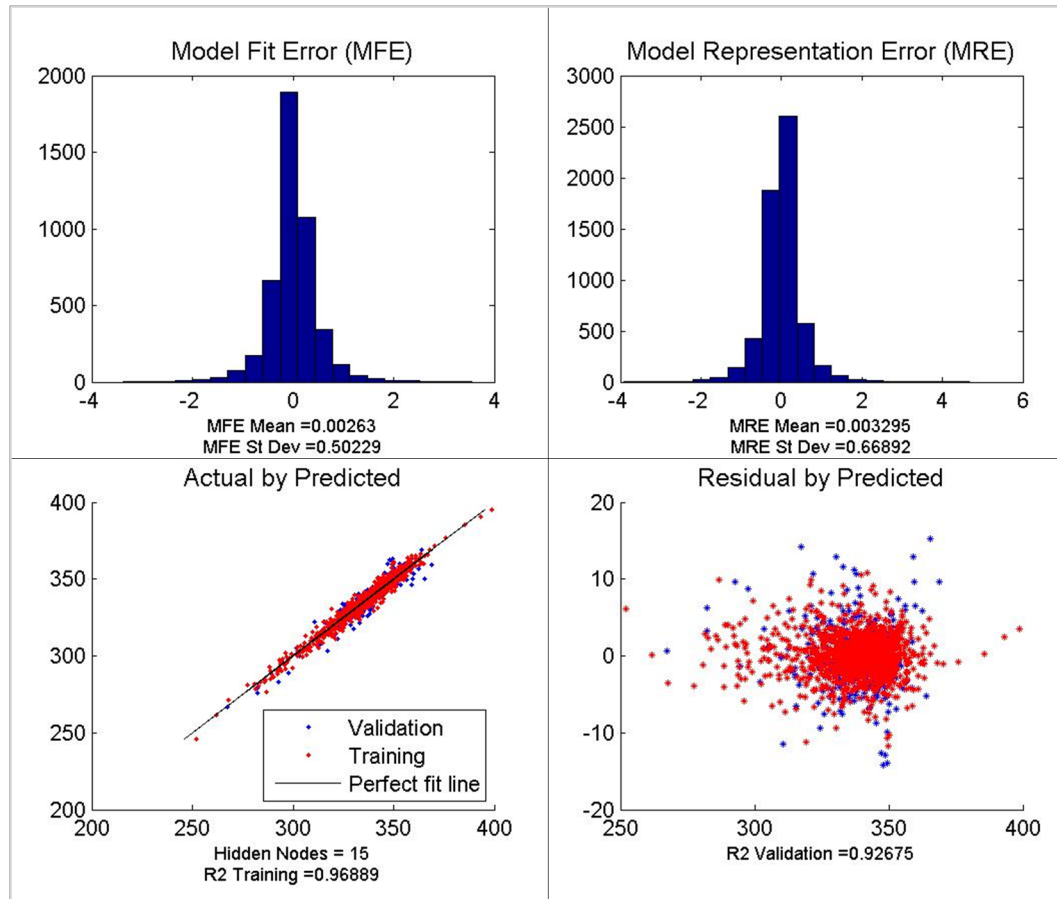


Figure 135: Goodness of Fit for Vtrend

Dynamic Response: VrippleInf

VrippleInf is the inf-envelope of the ripple signal, obtained after “denoising” the voltage responses with the MRA. The charts evaluating the goodness of fit of the output of the training, for 10 neurons in the hidden layer, are shown in Figure 136. As one can see in the relative error distributions, the maximum relative error reached is high. This phenomenon was encountered in Experiment 2: the signal takes values near 0, and small absolute regression errors may lead to extremely high relative errors.

The absolute error, shown in the residual vs. predicted plot, stays within reasonable bounds. Therefore, the fit is acceptable, though not excellent.

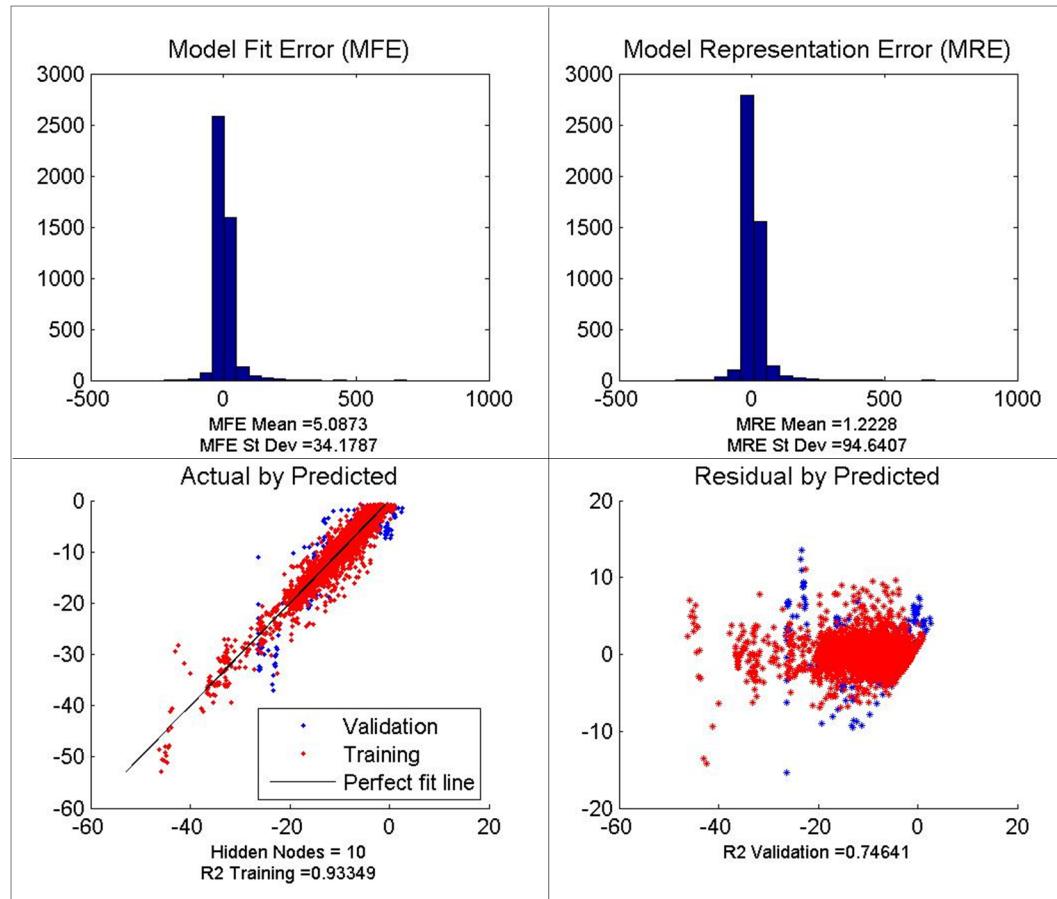


Figure 136: Goodness of Fit for VrippleInf

Dynamic Response: VrippleSup

VrippleSup is the sup-envelope of the ripple signal. The output neural network that approximates the behavior of VrippleSup contains 29 neurons in the hidden layer. The goodness of fit is evaluated with the charts shown in Figure 137. As with VrippleInf, the relative error reaches high values. However, the residual vs. predicted plot shows that the points are closely clustered around the zero error line.

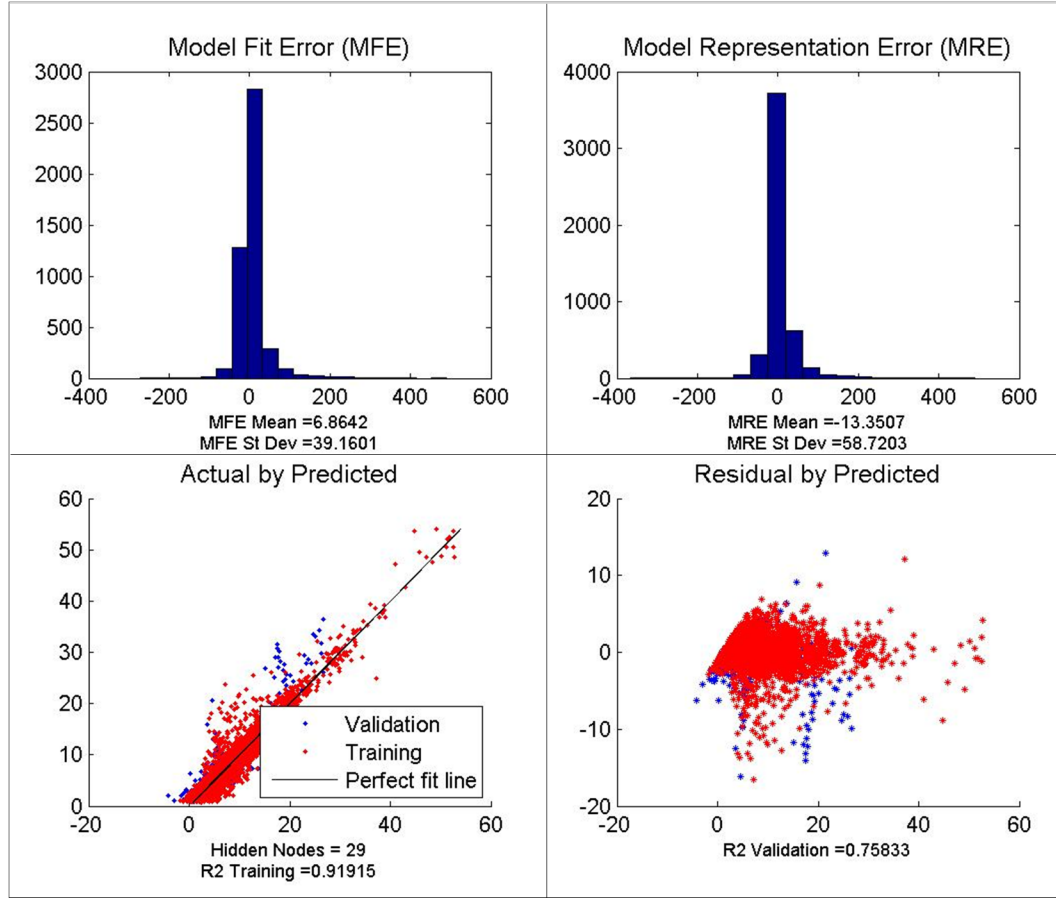


Figure 137: Goodness of Fit for VrippleSup

Summary

The fits, obtained after training the neural networks, are globally satisfactory, even though they are not excellent for the envelope signals. However, when recombined with the trend signal (with its values of higher order of magnitude), the absolute error will be “diluted” to yield low relative errors on the approximation of the global envelope (the envelope of the full multiscale voltage signal).

The output of the training process are sets of parameters of the resulting neural networks for each response. These parameters are composed of the connection weights, at the input of each neuron, as well as the “bias” parameters a and b for each neuron (cf. the generic neuron schematic of Figure 51).

One of the lessons learned from the training of neural networks is that despite its automation using commercial toolboxes and in-house software, there is still an element of trial-and-error, as the user can vary the initial number of nodes in the architecture and the optimization algorithm. Also, it was found advantageous to monitor the evolution of the regression errors during the training process since the user can stop the training if the errors reach satisfactory low values.

Training the neural networks thus takes some time and effort. In this experiment, an approximate average of 2 hours was needed to train each response. For the four neural networks that were trained, this amounted to a total of 8 hours of training. This is a significant time investment, but it enables the efficient design space exploration through the Monte-Carlo Simulation performed in the next step.

6.3.2.6 Step 6: Perform a Monte-Carlo Simulation Using the Dynamic Surrogate Models

Step 6 consists of running a Monte-Carlo Simulation (MCS) on the dynamic surrogate models generated in step 5 by training the neural networks. A total of 10,000 random settings for the design variables (thus forming 10,000 Monte-Carlo cases) was generated using the built-in function in the JMP software.

At the same time, the time interval was discretized from 0.95s to 1.2, in such a way that there were more samples in the “early transient” region. The “time vector” used in the MCS process was eventually composed of 48 time instants.

Thus, for each of the 10,000 random cases generated by the MCS, the neural networks were evaluated in Matlab for the static response Loss, and the dynamic responses corresponding to the trend V_{trend} , the ripple inf-envelope $VRippleInf$ and the ripple sup-envelope $VRippleSup$. The voltage inf-envelope and sup-envelope were obtained for all of the 10,000 cases by adding the V_{trend} and $VRippleInf$ on one hand, and V_{trend} and $VRippleSup$ on the other one. Therefore, the result of Step 6 is a set of data tables for each response, with 10,000 lines for Loss, and 48,000

lines (one line per time instant) for the dynamic responses V_{trend} , $VRippleInf$, and $VRippleSup$.

The generation of the Monte-Carlo case table took less than a second, and the total run time for the evaluation of the 10,000 Monte-Carlo cases took 13 seconds. This means that on average, a run of a Monte-Carlo case, including the simulation of 4 neural networks (for the responses loss, V_{trend} , $VRippleInf$ and $VRippleSup$), took 0.0013 seconds. Each simulation run of a single neural network therefore took approximately 0.3 milliseconds.

However, in order to rigorously assess the time savings yielded by surrogate modeling, one must take into account the time it took to generate the surrogate models as listed in the previous steps. It was seen that the test planning took 17 minutes, that the test campaign took 2 hours and 25 minutes, and that the training of the four neural networks took 8 hours. Therefore, the total time needed to generate and evaluate the 10,000 Monte-Carlo cases was 10 hours and 42 min.

Without surrogate modeling, the 10,000 Monte-Carlo simulation runs would have been performed on the original Time-Domain Simulation model in Simulink. It was seen during the test campaign that on average, a run took 34 seconds, which is about 10,000 times longer than the simulation run of a neural network. Thus, without surrogate modeling, the Monte-Carlo simulation, needed to populate the design space visualization and exploration VisTRE, would have taken $10,000 * 34 \text{ sec} = 340,000 \text{ sec}$, i.e. 94 hours and 26 minutes.

Therefore, static and dynamic surrogate modeling enabled a time savings of 84 hours. Moreover, the generation of surrogate models now enables a vast number of additional Monte-Carlo to run in a matter of seconds. This could be desirable if one wishes to refine the design space sampling or focus on a particular region of the design space. Therefore, the time invested in training the neural networks pays off, as mentioned earlier.

6.3.2.7 Step 7: Populate the Interactive Visualization Environment

In step 7, the data tables generated by Step 6 were formatted and manipulated in the manner described in the previous experiment (describing the use of VisTRE). Thus, the data and signals produced by the Monte-Carlo Simulation could be imported into JMP, and were ready for visual mining through VisTRE.

Then, the visualization environment VisTRE was created and populated by calling the script “Create_VisTRE” described in the previous experiment. The scatterplot for the static responses and the overlay plots for the envelope signals (V_{inf} and V_{sup}) of the dynamic response V were thus created, as illustrated in Figures 138 and 139. One can see that the discretization of time is apparent in the overlay.

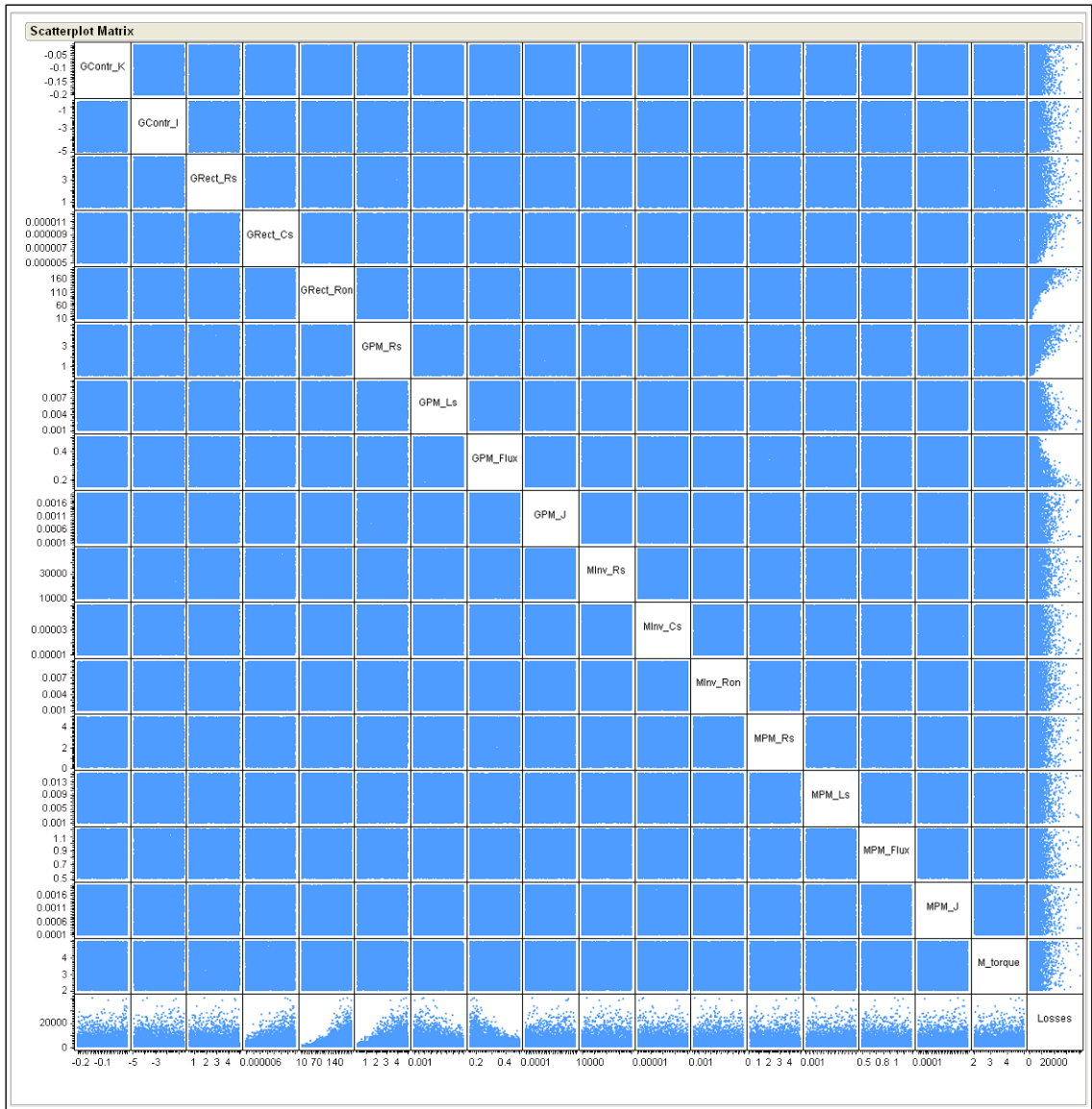


Figure 138: VisTRE: Scatterplot Matrix for Design Variables and Loss

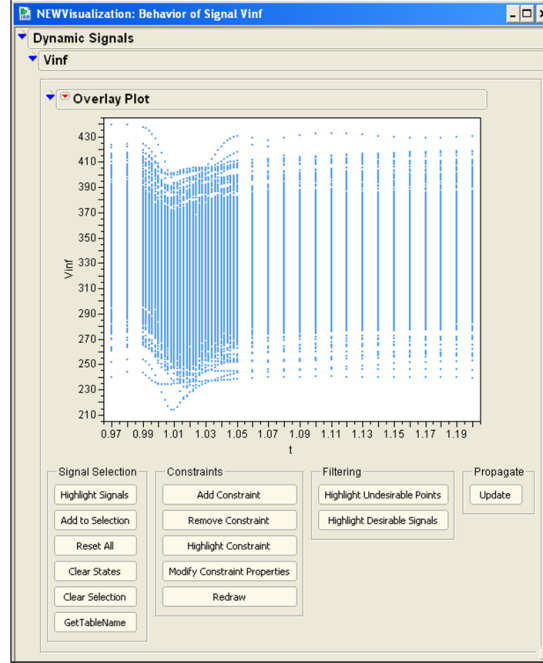


Figure 139: VisTRE: Overlay Plot for Vinf

6.3.2.8 Step 8: Explore and Filter the Design/Operation Space

Now that the VisTRE environment is populated, the design space can be explored and filtered, as explained in chapter V. Because the transient perturbation under consideration induces an undervoltage, only the results pertaining to Vinf are made explicit here.

The interactivity of VisTRE enables the user to quickly query the design space. For instance, on the scatterplot, the user can select the four points that show the highest values for the electrical loss, as illustrated in Figure 140. Then the selected design points can be propagated to the overlay in order to visualize the corresponding behavior for the Vinf signal, as seen in Figure 141.

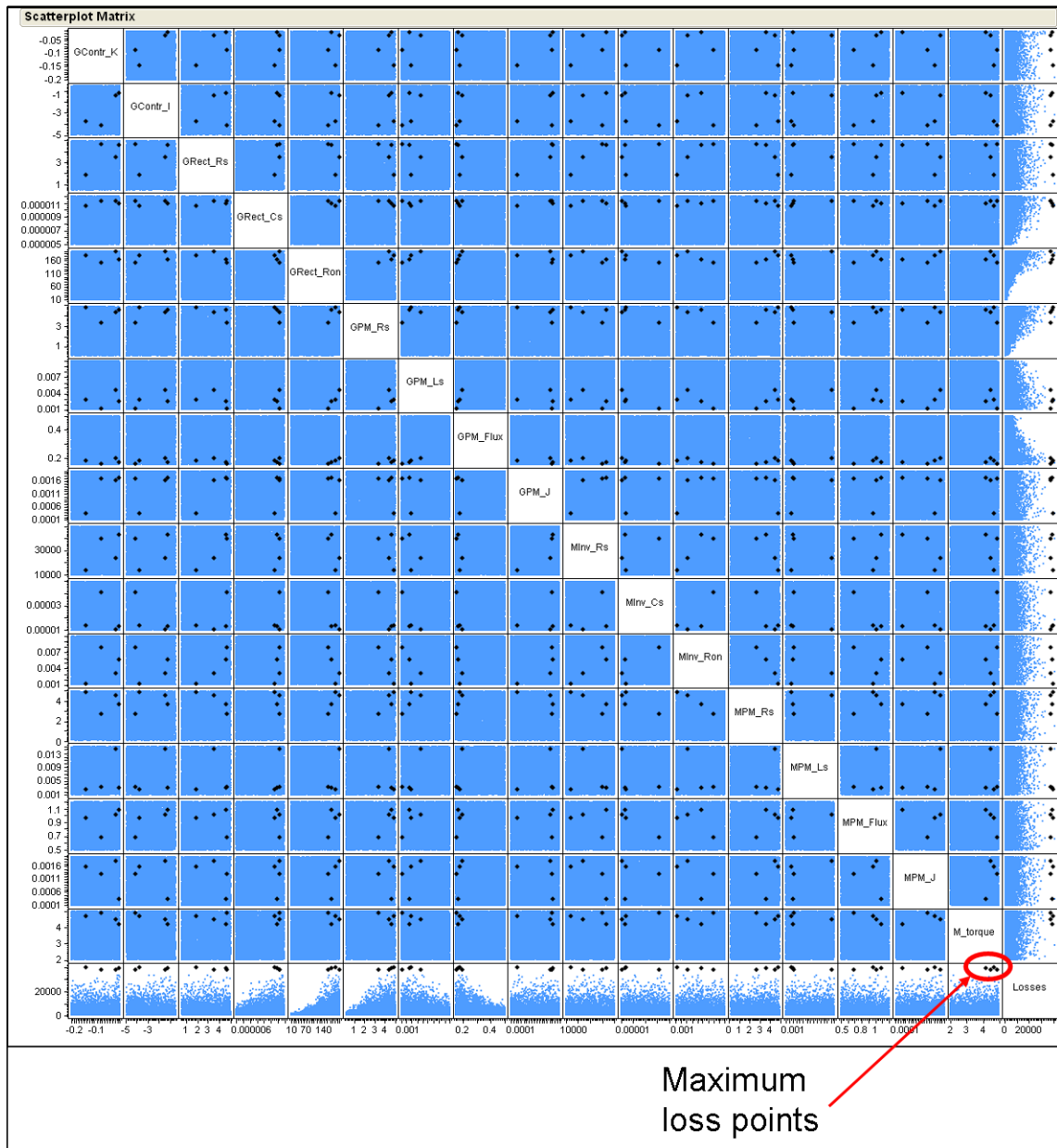


Figure 140: VisTRE: Maximum Loss Points

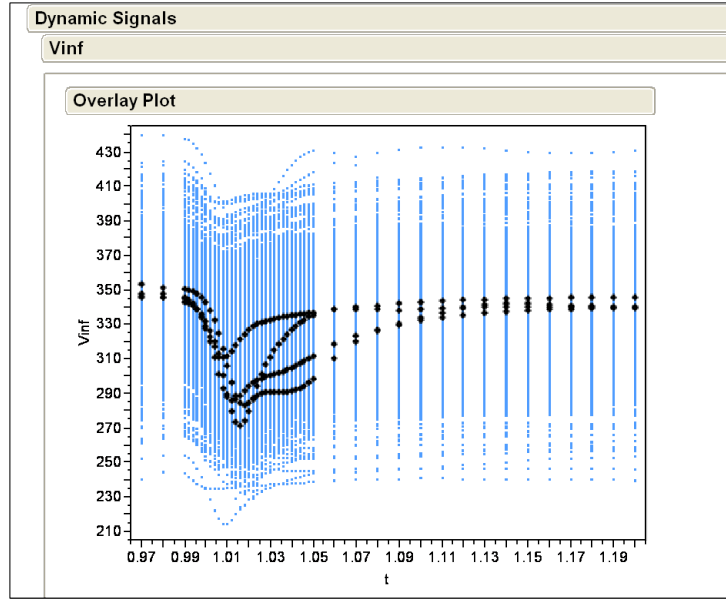


Figure 141: VisTRE: Vinf Behavior for the 4 Maximum Loss Points

In order to filter the points that violate the transient constraint, the user first needs to load the dynamic constraint into VisTRE. The lower dynamic constraint of the transient power quality constraint is added to the overlay for Vinf, the inf-envelope of voltage. This is illustrated in Figure 118.

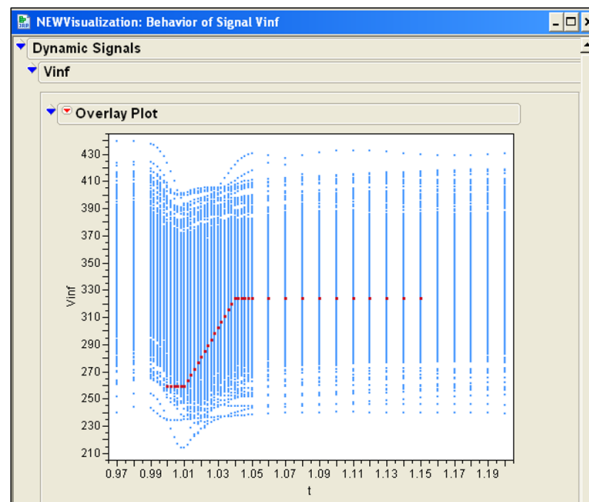


Figure 142: VisTRE: Overlay of Vinf with Minimum Constraint

As one can see on the figure above, a substantial portion of the overlay plot falls below the dynamic constraint in red. In order to select this portion, the appropriate script is called by pressing the action button “Highlight Undesirable Points”. Then the full signals corresponding to the “bad” portion of the overlay can be highlighted and selected. The results are shown in Figure 143, where only the time region violating the constraint is highlighted in black, and in Figure 144, where the full signals are selected and highlighted in black. Thus, all the signals that at some point in time fall below the constraints are selected.

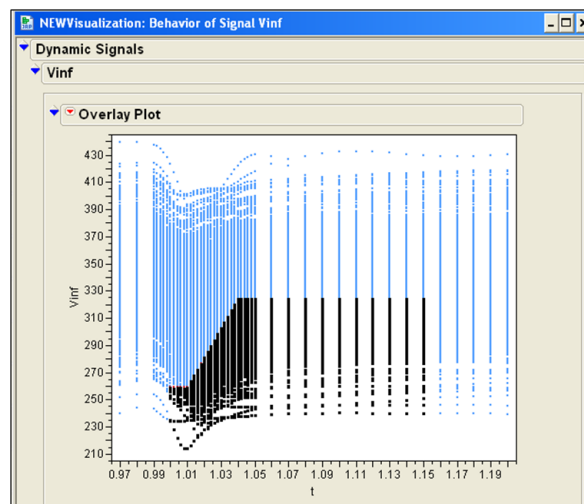


Figure 143: VisTRE: Finding the Points Violating the Constraint in the Vinf Overlay

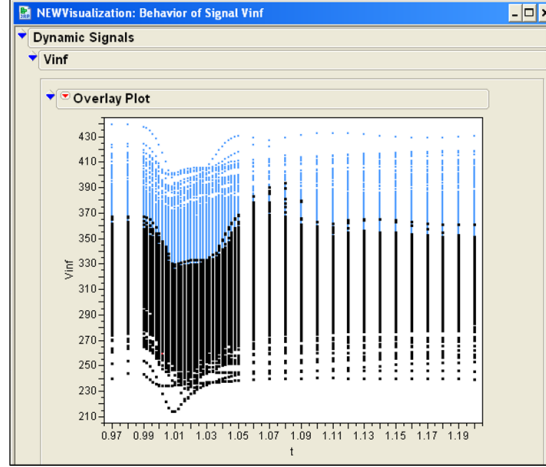


Figure 144: VisTRE: Selecting the Full Signals Violating the Constraints in the Vinf Overlay

The results of the filtering process can be propagated to the scatterplot, in order to visualize the design points that violate the dynamic constraints in the overlay plot of Vinf. This is shown in Figure 145, where the undesirable points are highlighted in black. One can see that a significant portion of the design space is covered in black. However, a scatterplot completely covered in black does not necessarily imply that the entire design space is covered in black. This is due to the fact that the scatterplot represents 2-D projections on 2-D planes of a higher dimension space. For instance, if the design space is a 3-D black cube that contains a blue ball inside, the scatterplot will show black points scattered as if they covered the entire design space. Out of the 10,000 Monte-Carlo cases, 2016 were found to violate the transient constraint for Vinf and highlighted in black, which thus represented more than 20% of the design space.

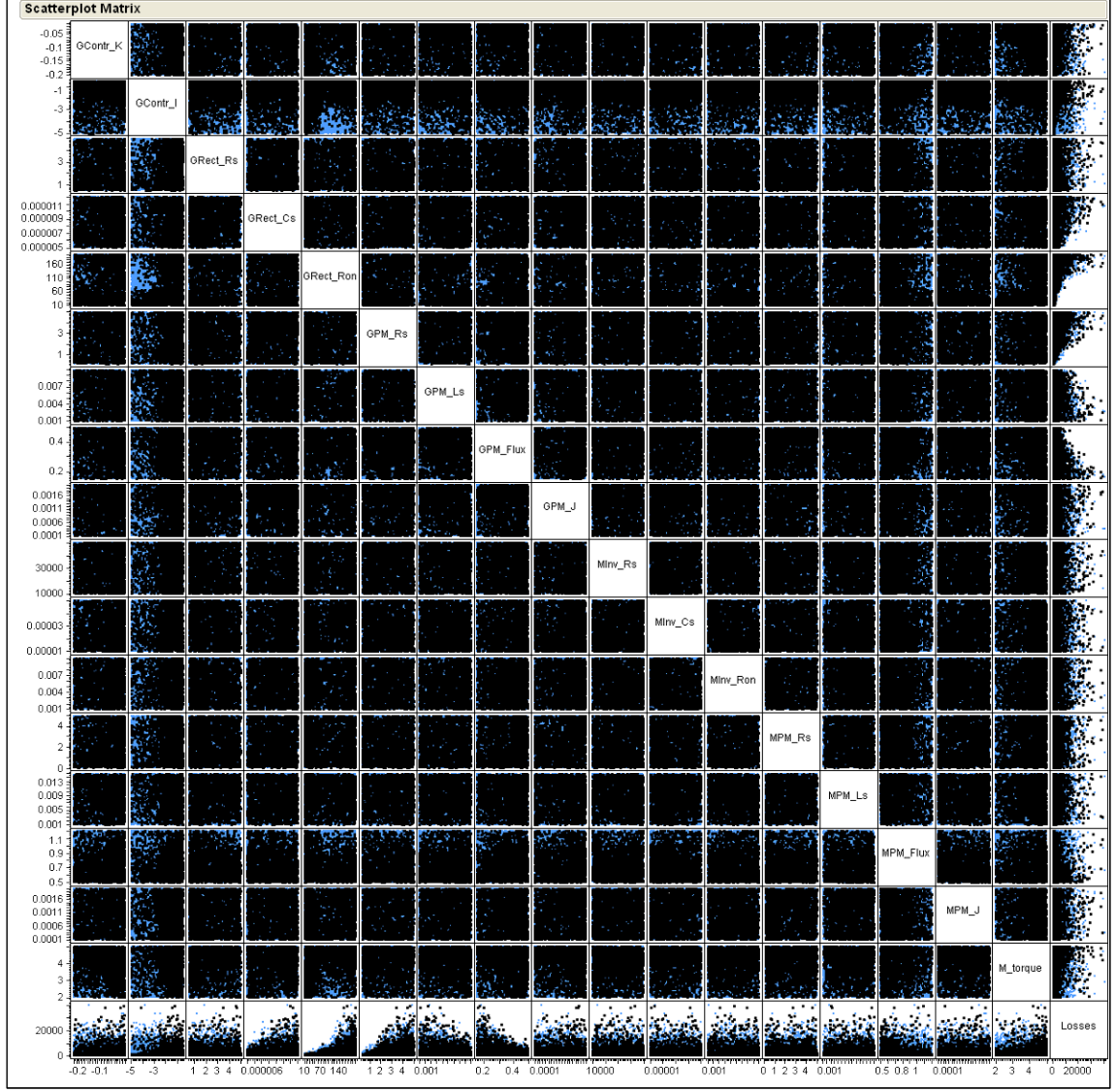


Figure 145: VisTRE: Filtered Scatterplot

In the previous scatterplot, one can notice continuous regions that are entirely blue, especially in the cell plotting the controller integral gain (GContr_I) against the generator rectifier internal resistance (GRect_Ron). The presence of this region, identified as the “region of comfort” in Figure 146, means that choosing a design point in this region ensures the designer that the system will meet the transient constraint, regardless of the values of the other design variables.

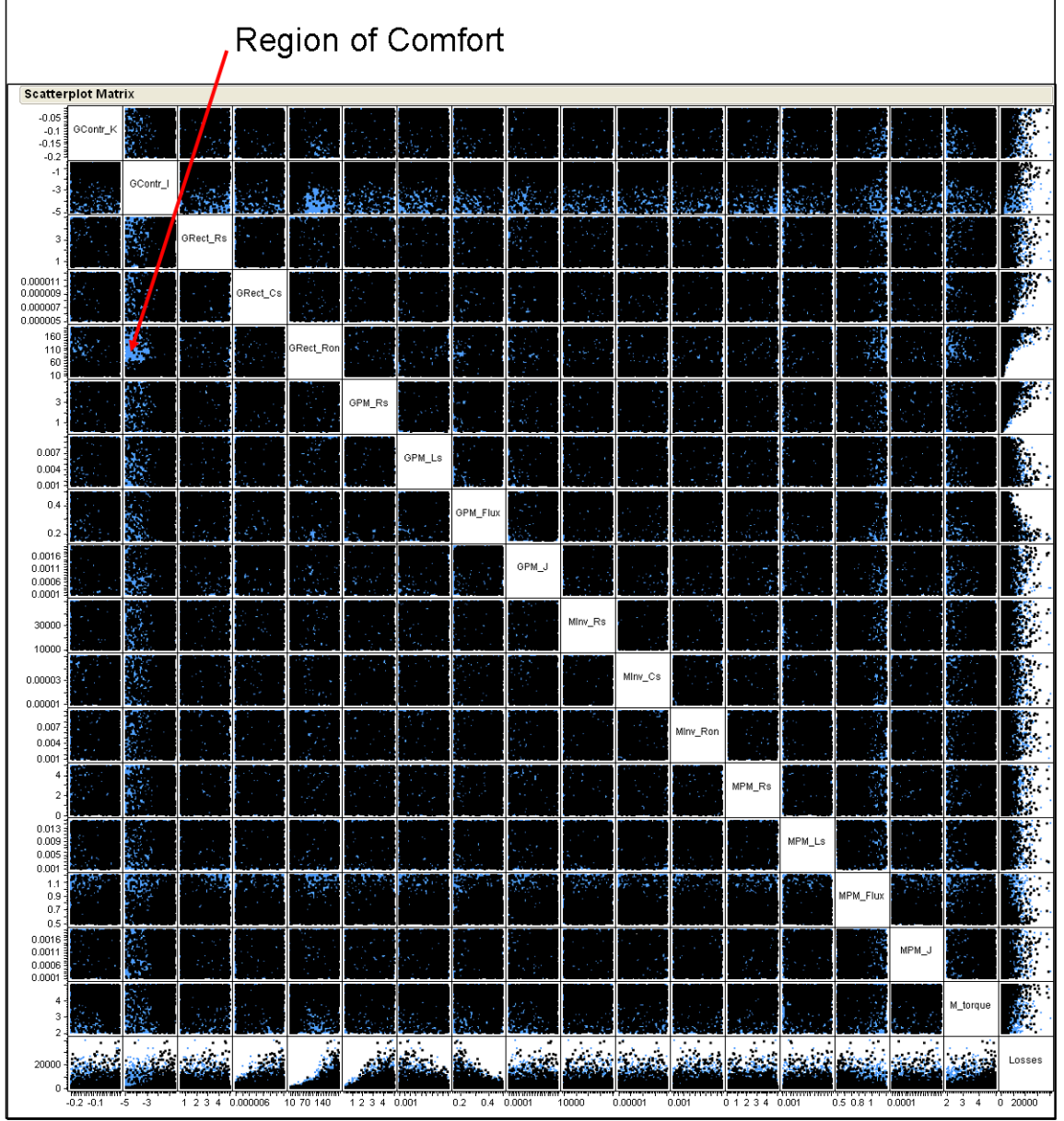


Figure 146: VisTRE: Filtered Scatterplot with Region of Comfort

The region of comfort, corresponding to values of $GContr_I$ and $GRect_Ron$ defined in Equation 49, is highlighted in purple in Figure 147, where the axis of the last row (corresponding to the electrical loss) has been adjusted in order to “zoom in” and visualize the region of minimum losses.

$$-5 \leq GContr_I \leq -4.35 \text{ and } 79 \leq GRect_Ron \leq 135 \quad (49)$$

Without the interactive visualization environment VisTRE, the designer might have chosen design points within the region of comfort. However, one can see in the scatterplot that the region of comfort can achieve, at best losses around 50W, while the scatterplot shows that there are points in blue (which don't violate the transient constraints) that can achieve losses with values below 30W.

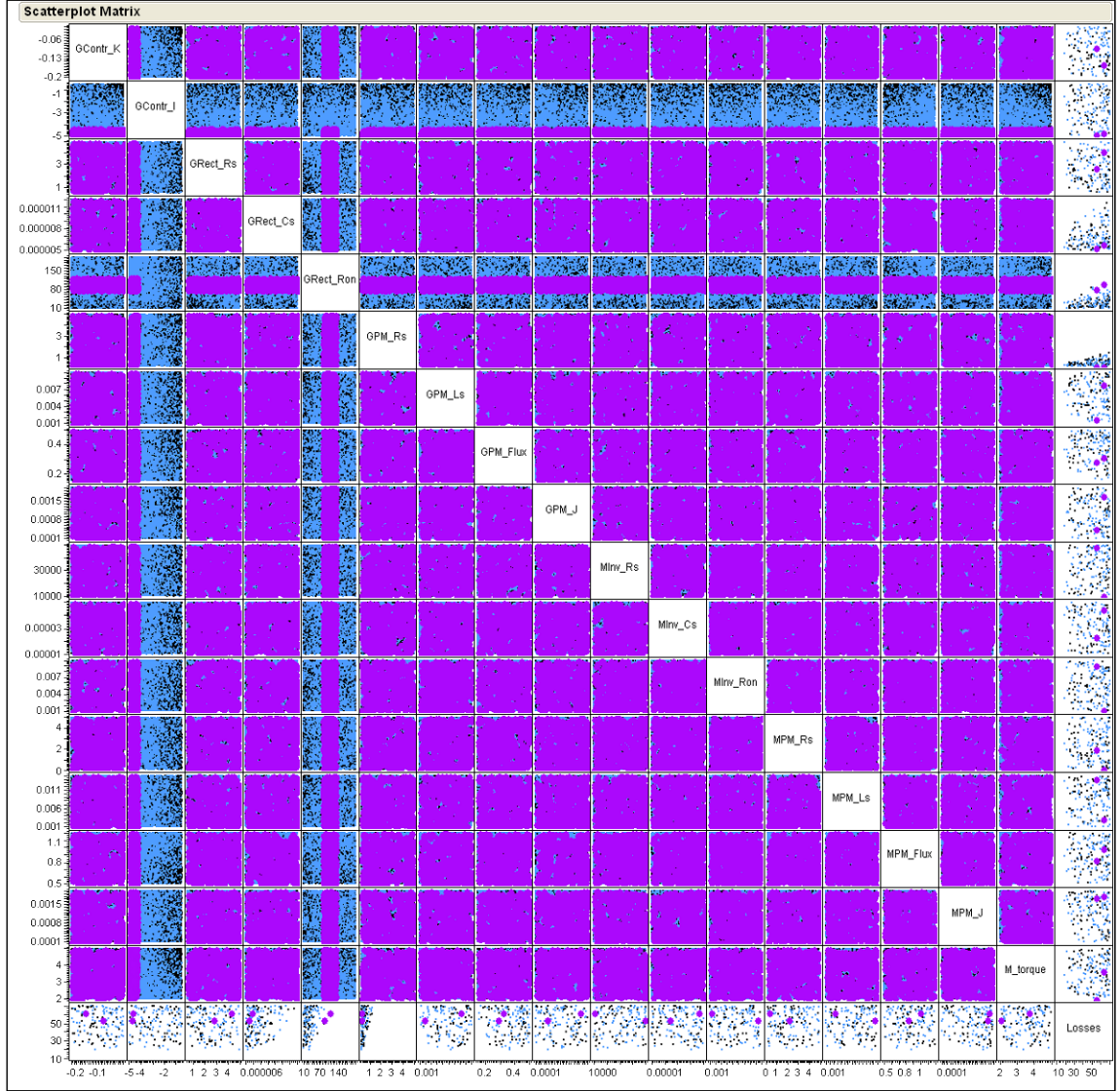


Figure 147: VisTRE: Filtered Scatterplot with Highlighted Region of Comfort

In order to identify the most desirable design points among those that meet the transient constraints, the regions that violate the dynamic transient constraints can

first be “filtered out” by eliminating them in the scatterplot and in the overlays. The resulting filtered design space is visualized in Figures 148 and 149. One can see in the scatterplot that an important portion of the design space remains.

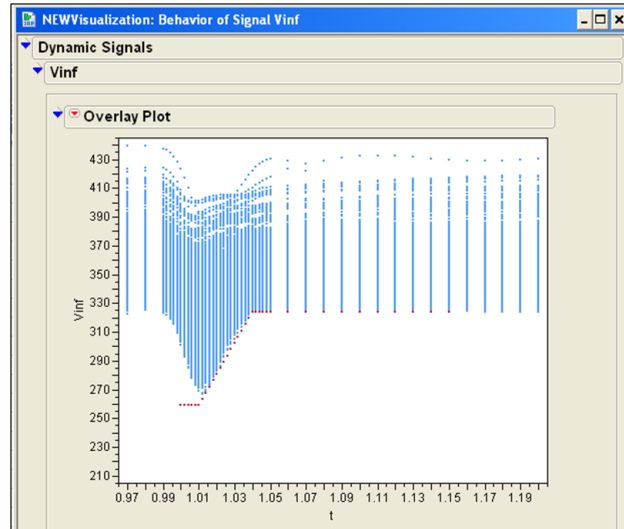


Figure 148: VisTRE: Filtered Overlay with Hidden Undesirable Region for Vinf

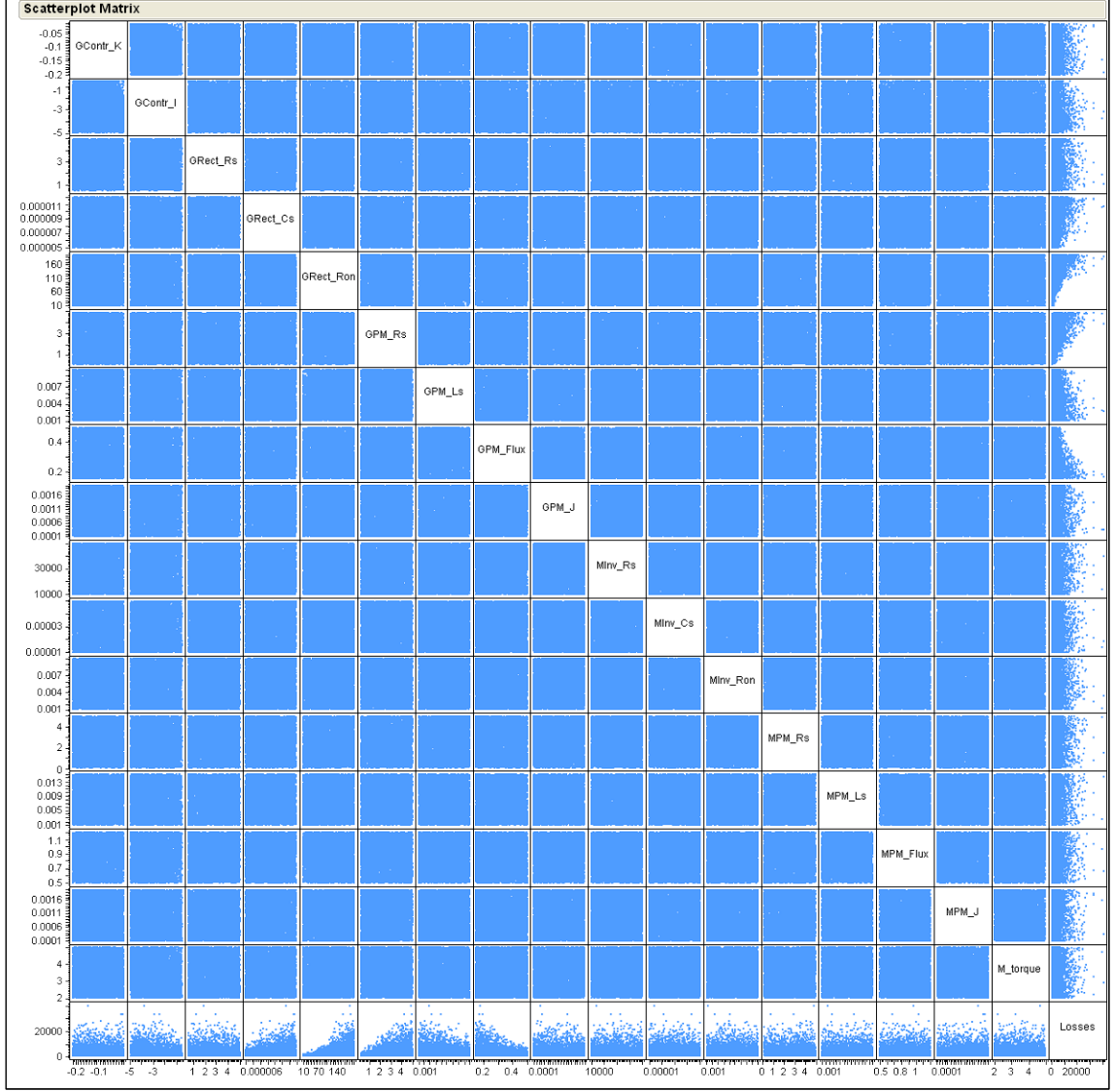


Figure 149: VisTRE: Filtered Scatterplot with Hidden Undesirable Region for Vinf

At this stage, only the 7,984 design points (out of the original 10,000) remain, corresponding to the design points that meet the transient constraint for Vinf. From this point, further filtering can be applied on the electrical losses (response “Losses”) in order to reduce the design space to the minimal loss region. For example, one can select, among the remaining points, those corresponding to values of loss lower than 30W, as shown in Figure 150. There were eight design points left, and the interactivity

of the environment in JMP then allowed to create a subset of the original data table in order to visualize the values of the design variables for these remaining points.

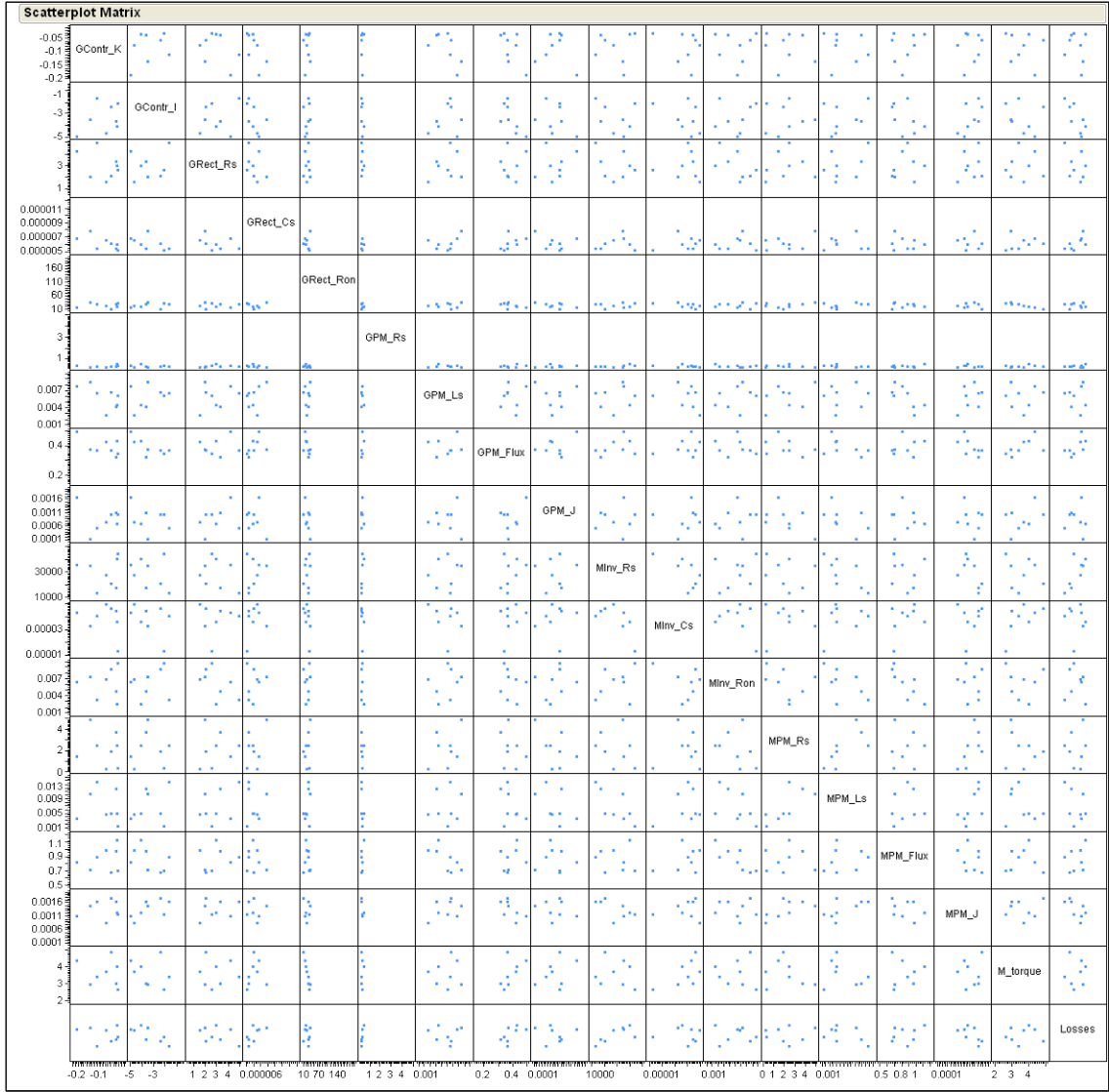


Figure 150: VisTRE: Filtered Scatterplot with Minimum Loss Points

6.3.3 Summary and Methodology Validation

The methodology formulated in this thesis was successfully applied for the design space exploration of the 350VDC electrical network. The design space was iteratively trimmed down in order to keep only the design points that do not violate the constraints.

In order to fully validate the methodology, it is now compared to the results of the traditional Filtered-Monte-Carlo, where only the static parameters are considered. For this purpose a maximum constraint of 200W is applied on the Loss response of the scatterplot matrix. The results of the corresponding filtering is visualized in Figure 151.

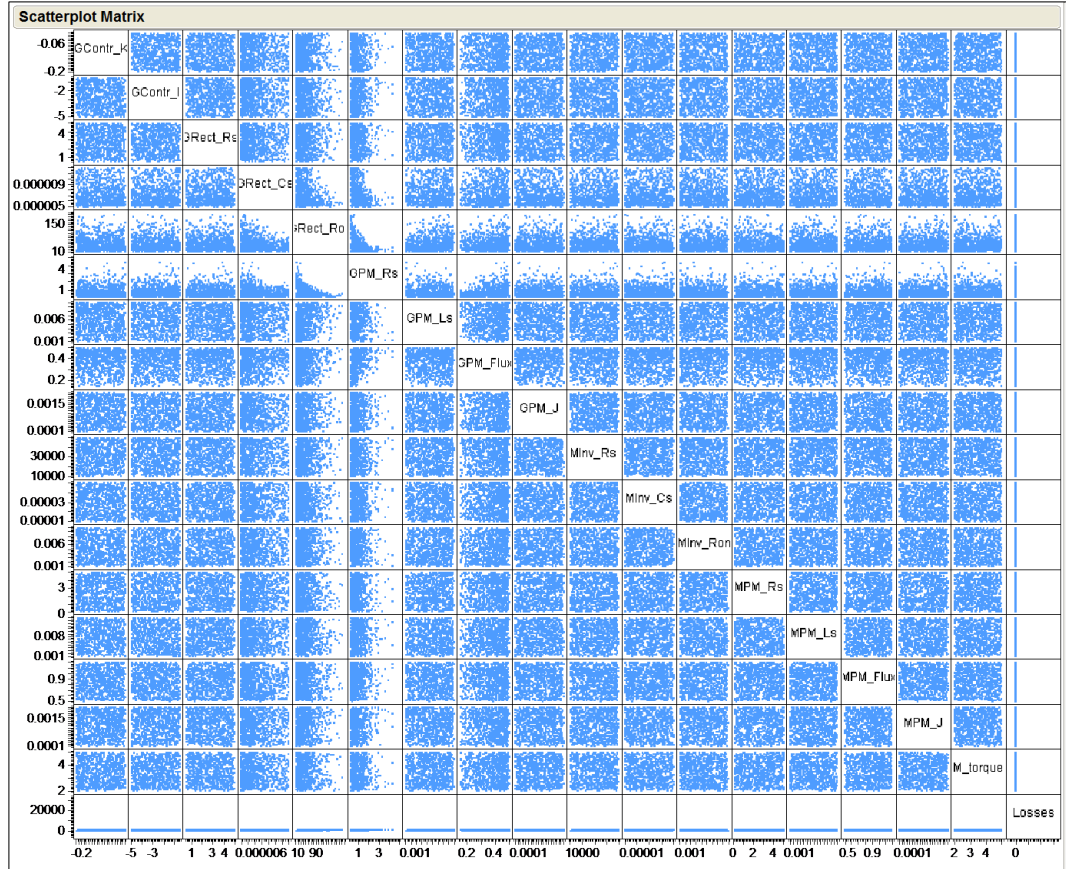


Figure 151: Loss Constraint in the Scatterplot for the Original Filtered-Monte-Carlo Method

Now, the same constraint for the electrical loss ($Loss \leq 200W$) is applied to the filtered scatterplot of Figure 145 obtained in the previous paragraph, after applying the Time-Domain Filtered-Monte-Carlo enabled by VisTRE. The resulting filtered scatterplot is shown in Figure 152, where the points that violate the dynamic transient constraint for the lower envelope of the network voltage appear in black.

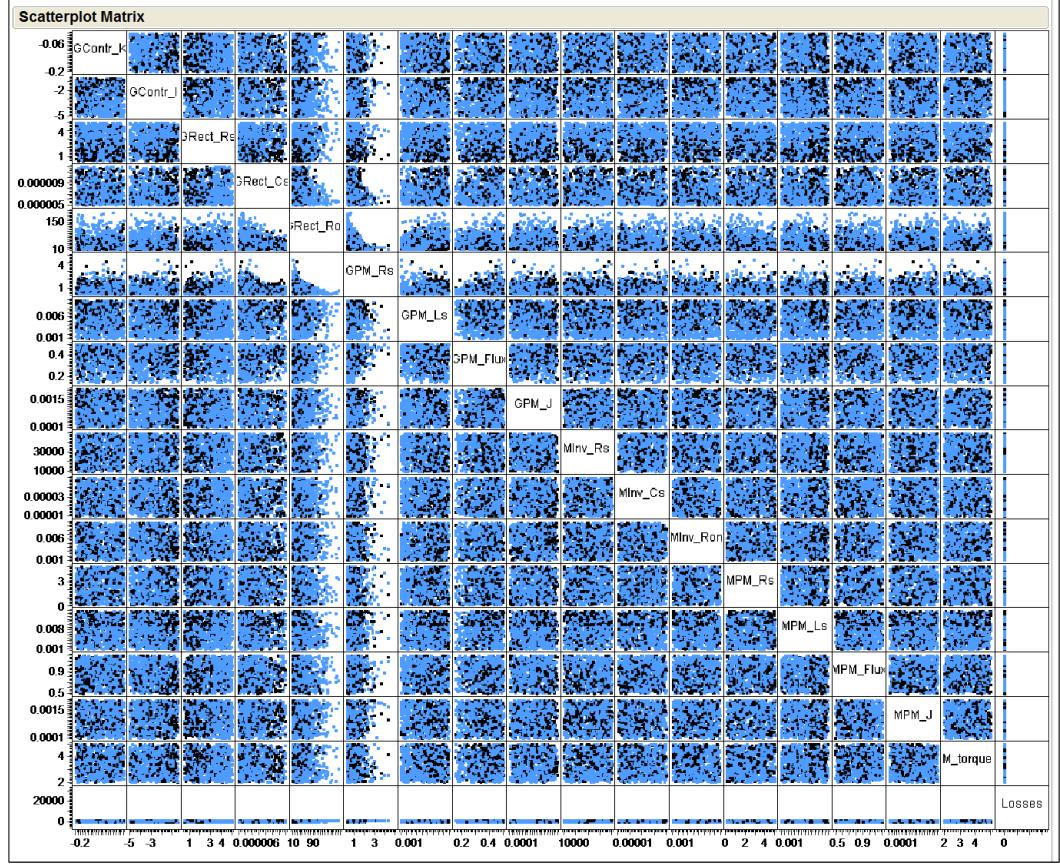


Figure 152: Loss Constraint in the VisTRE Scatterplot for the Time-Domain Filtered-Monte-Carlo

One can see in the VisTRE scatterplot that the region of minimum electrical losses contains design points (in black) that violate the dynamic constraints for transient power quality. Therefore, if done without taking into account the transient constraints as formulated in the methodology of this thesis, the design space exploration would have carried design points that might later reveal themselves as violating these constraints, which would have induced the type of design reiterations described in Chapter I. In a nutshell, the methodology formulated in this thesis does enable a more efficient exploration of the design space of a dynamic system that is subject to dynamic transient constraints.

Chapter VII

CONCLUDING REMARKS

7.1 Recapitulation

Throughout the mission, aircraft dynamic systems experience transient regimes that can severely affect the good operation of the aircraft. In order to ensure that transient behaviors do not have serious consequences, the transient responses of dynamic systems to expected or unexpected perturbations are often regulated by dynamic constraints, so called because their values vary with time. The verification of these constraints generally necessitates time-domain simulation, which intervenes late in the specification of systems, thus potentially inducing costly design iterations.

A design methodology has been formulated in which transient dynamic constraints can be efficiently evaluated while optimizing the system. As a building step for this methodology, dynamic surrogate models of the original parametric time-domain simulation models are generated in order to speed up the simulation process. To generate these dynamic surrogate models, this thesis proposes to follow a nonlinear system identification approach based on multivariate wavelet networks. These are built around the theory of wavelet decomposition, which has become popular for its ability to capture the multiscale nature of transient signals.

However, as shown in Experiment 1, training wavenets becomes computationally prohibitive with the optimization learning algorithm used in this thesis. Therefore, an alternate approach was formulated, in which the envelope of the dynamic response is extracted and regressed with traditional sigmoid-based feedforward neural networks. The extraction of the envelope is performed via a wavelet-based MultiResolution Analysis, which separates time-domain signals into an approximation signal

(the trend) and a detail signal (the ripple or the noise). A sliding windowing method is applied to the ripple signals in order to compute the envelopes.

The efficiency of the resulting dynamic surrogate models enable the implementation of a data farming approach, in which a Monte-Carlo simulation is run to generate a massive set of scenarios spanning the vast design and operation space. In order to perform visual mining of the transient responses of the dynamic system, an interactive visualization environment called the Visual Transient Response Explorer (VisTRE) was developed. VisTRE enables visual queries of the design space and interactive what-if analyses. The user can thereby instantaneously comprehend trends of the transient response of the system, and its sensitivities to the design and operation variables. This multivariate environment includes the visualization of the static and dynamic responses, and provides the designer with the ability to instantly add constraints and to filter the design space to have it exhibit only the design scenarios verifying the dynamic and static constraints. Thus, the user can pursue the optimization process on a design and operation region that is sure to meet the static and dynamic transient constraints.

A series of experiments was performed in order to incrementally test the hypotheses and the methodology. First, the wavenet system identification approach was tested, and it was shown that beyond a number of 2 design variables, the training process becomes unmanageable. Then, the visual environment VisTRE was created and tested on a simple case. Finally, the full methodology was tested for the design space exploration of a 350VDC electrical network. The aim of the design activity was to minimize the electrical losses incurred by the controlled generator while ensuring that the network voltage did not violate the transient power quality constraints, which were dynamic. By comparing the process to the outcome of the traditional

Filtered-Monte-Carlo, which does not take into account dynamic constraints on dynamic responses, it was shown that the formulated Time-Domain Filtered-Monte-Carlo methodology can be essential in order to reduce the risk of design iterations.

The methodology was derived after following a thought process derived from the scientific method. After observations were made on the problem at hand, the research objective was refined. This prompted a series of research questions and hypotheses. The objectives, the research questions and the hypotheses are recalled once more below, and the methodology is recalled in Figure 153.

Research Objective 1: Develop a methodology that integrates transient regime analysis and pertaining dynamic constraints into the design synthesis of aircraft dynamic systems.

Research Question 1 (RQ1): How can one efficiently and thoroughly explore the design and operation spaces while accounting for dynamic responses and pertaining uncertain dynamic constraints?

Hypothesis 1 (H1): A data farming approach, based on the integration of time as a dimension in the the Filtered-Monte-Carlo approach, will conduce to the efficient and thorough exploration of the design/operation space for dynamic signals with dynamic constraints.

Research Question 2 (RQ2): How can one make the system simulation process more efficient in order to speed up the optimization/verification of dynamic systems and facilitate the implementation of the proposed Time-Domain Filtered Monte-Carlo technique?

Hypothesis 2 (H2): Dynamic surrogate modeling of time domain simulation models will enable the efficient implementation of the Time-Domain Filtered Monte-Carlo technique for the exploration of design/operation space of dynamic systems subject to dynamic constraints.

Research Question 3 (RQ3): What system identification approach is suitable for the generation of dynamic surrogate models in the context of transient time domain simulation of dynamic systems subject to dynamic constraints?

Hypothesis 3a (H3a): A nonlinear system identification approach based on wavelet neural networks will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems.

Hypothesis 3b (H3b): A nonlinear system identification on the envelope of the signals, using sigmoid-based feedforward neural networks, will enable the generation of dynamic surrogate models of the transient behavior of dynamic systems, and the implementation of the Time-Domain Filtered-Monte-Carlo.

7.2 *Contributions*

While formulating and implementing this thesis, several contributions were made. The most important one was the formulation of a methodology enabling the designer to integrate the verification of dynamic transient constraints into the early phases of the specification of dynamic systems. Instead of relying on trial-and-error or rules-of-thumb, the methodology presents a systematic way of ensuring that transient dynamic constraints are met.

Then, a framework for the generation of dynamic surrogate models was formulated. This framework includes the wavenet approach as well as the approach that integrates time as an input parameter of traditional sigmoid-based feedforward neural network. Initial guidelines for the applicability of wavenets to the generation of multivariate dynamic surrogate models were derived. In order to implement the envelope approach, an simple envelope extraction scheme was formulated. This scheme

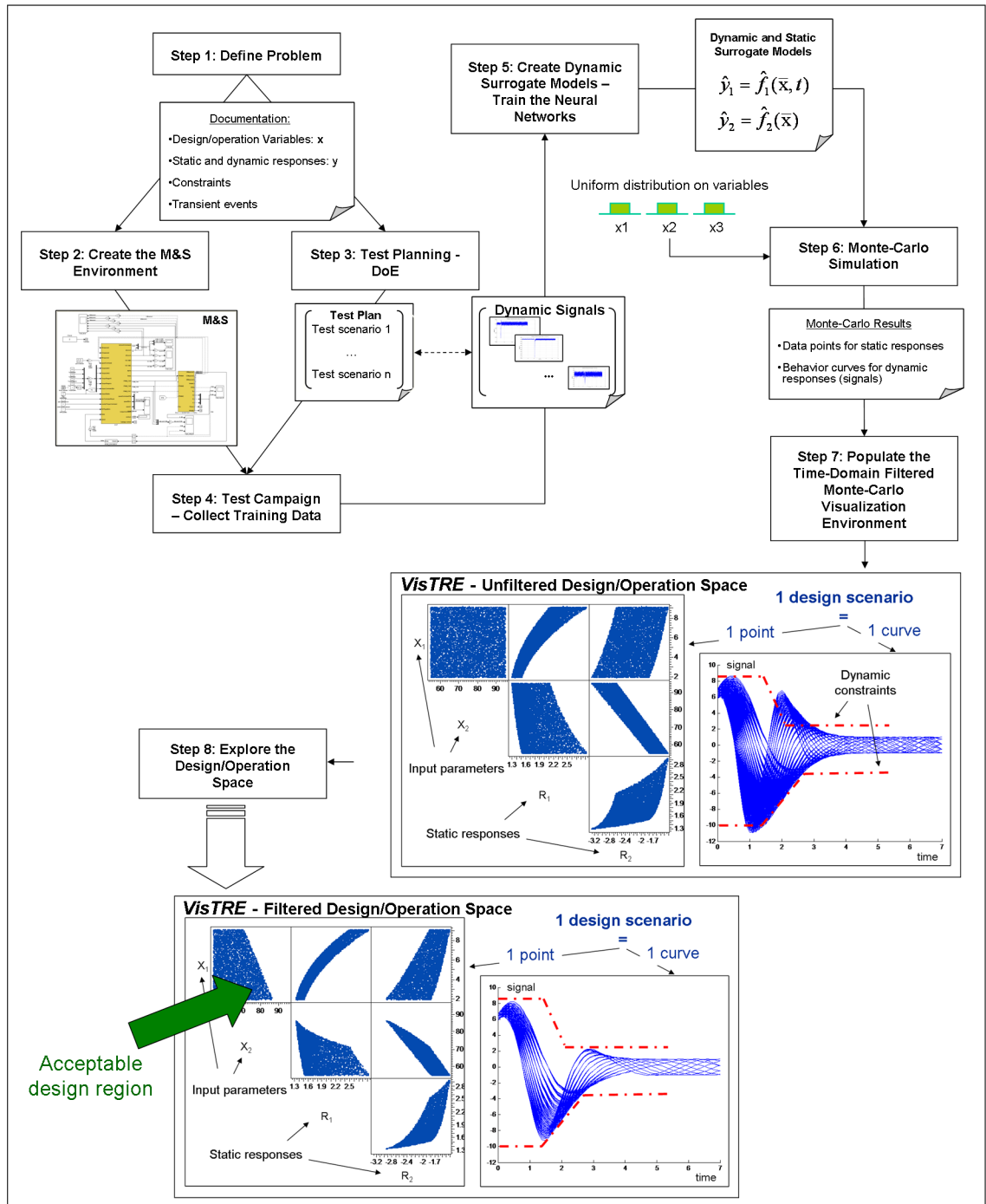


Figure 153: Methodology Overview

was shown to handle the multiscale aspect of time-domain signals by separating them through a wavelet-based MultiResolution Analysis (MRA) approach.

Moreover, a framework was derived for the evaluation of the goodness of fit of dynamic surrogate models. Indeed, it was seen that the time dimension has peculiarities that necessitated adaptation of the traditional goodness of fit methods. Also, a framework for the application of Monte-Carlo simulation to time-domain simulation models was created. For this activity, reusable Matlab templates were generated.

It was also shown that the methodology enabled the designer to concurrently and efficiently design dynamic systems and their controllers. If the methodology is applied to more complex controllers, this is a potentially highly rewarding achievement, since it was seen in Chapter I that the tuning of the controller parameters can be a tedious and costly trial-and-error process.

Finally, a visual interactive environment, VisTRE, was created in JMP for the visual mining of design spaces that include dynamic responses. For this purpose, a set of JMP template tables and scripts were created.

7.3 Limitations and Recommendations for Future Work

The methodology and its implementation presented some limitations, which can be regarded as possible paths of research in future work. First, it was seen that the wavenet approach, as implemented in this thesis, stalled when training networks composed of more than three input parameters (including time). For higher numbers of design variables, the approach based on envelope identification needs to be implemented. However, working on the envelope loses information on the signal. Thus, the methodology would greatly benefit from the investigation of more efficient optimization algorithms for training. Also, a more intelligent way to initialize the parameters of the wavenet architecture (number of wavelons, initial weights, initial

dilation and translation factors) could be helpful in the quest for more efficient and robust training.

Another path for improving the methodology pertains to the envelope extraction method. In this thesis, a sliding windowing method was used to extract the envelope of the ripple signal (noise around the trend signal). The window length used in the sliding windowing was chosen after a trial and error process. A more systematic set of rules is needed for the choice of the windowing parameter and the level of resolution in the MRA wavelet decomposition. The sensitivity of the quality of the extracted envelope to these parameters should be studied.

In the implementation of the thesis methodology, the assignment of ranges to the design variables was done arbitrarily. Typically, expert opinion is needed for this step. For the controller parameters (gains), a few trial-and-error runs were performed to ensure that the signal remained stable at the extremes of the controller parameter ranges. A more systematic, efficient, and intelligent way of determining these ranges would be highly beneficial. For instance, one could envisage the integration of traditional analysis tools of control theory, such as methods based on eigenvalues and the root locus, which could be implemented as extensions of the static scatterplot.

Also, the first chapter suggested that the design of controlled dynamic systems could be made more efficient by performing a concurrent design of the controlled plants and their controllers. The concept, which fell beyond the scope of the present thesis, was successfully tested in the last experiment for a simple controller architecture. Future research could thrive to develop robust methodologies for the concurrent design of plants and controllers. For instance, the controller architecture could be treated as a design variable (which would be discrete), or more complex architectures could be investigated.

Finally, it was seen in the first chapter that uncertainty is present throughout the development of complex systems. The interactivity of VisTRE coupled with the full

exploration of the design space enables the designer to quickly study various constraint scenarios. However, the methodology developed in this thesis does not provide a systematic means of explicitly accounting for uncertainty. In a future work, one may seek to extend the methodology so as to systematically and efficiently integrate the effects of uncertainty on design variables and on constraints.

Appendix A

WAVELET THEORY: BACKGROUND

This section gives a formal introduction to wavelet theory, the continuous wavelet transform, the discrete wavelet transform, which form the basis of the wavelet decomposition and the multiresolution analysis theories described in chapters III and IV. The background presented here is adapted from Mallat [102], Daubechies [35] and Teolis [145].

A.1 Mother Wavelet and Daughter Wavelet

In most cases, mother wavelet ψ are chosen to be continuously differentiable functions, in $L^1(\mathbb{R}) \cap L^2(\mathbb{R})$. This allows to formulate conditions of zero mean and square norm one:

$$\begin{aligned}\int_{-\infty}^{\infty} \psi(t) dt &= 0 \\ \int_{-\infty}^{\infty} |\psi(t)|^2 dt &= 1\end{aligned}$$

From a mother wavelet ψ , daughter wavelets $\psi_{a,b}$ can be defined by scaling (with scaling parameter $a > 0$) and translation (with translation parameter $b \in \mathbb{R}$):

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right)$$

A.2 Wavelet Transforms

Given a mother wavelet ψ , the Continuous Wavelet Transform (CWT) of a signal $x(t)$ in $L^1(\mathbb{R})$ for a scaling parameter a and a translation parameter b is defined as:

$$X(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t) \cdot \psi^* \left(\frac{t-b}{a} \right) dt$$

where the operator $*$ designates the complex conjugate.

In the continuous wavelet transform, the scaling parameter and the translation parameter are allowed to take continuous values. If one discretizes the values taken by a and b into a lattice formed by the set $\{(a^m, na^mb) : m, n \in \mathbb{Z}\}$, the wavelet transform becomes the Discrete Wavelet Transform (DWT). The daughter wavelets are then defined as:

$$\psi_{m,n}(t) = a^{-m/2} \psi(a^{-m}t - nb)$$

In most practical cases, $a = 2$ and $b = 1$ so that the daughter wavelets form a dyadic lattice:

$$\psi_{m,n}(t) = 2^{-m/2} \psi(2^{-m}t - n)$$

There exists numerous mother wavelets ψ for which the family $\{\psi_{m,n} : m, n \in \mathbb{Z}\}$ forms a basis of $L^2(\mathbb{R})$. In this case, the signal $x(t)$ can be reconstructed using the Inverse Discrete Wavelet Transform (IDWT):

$$x(t) = \sum_{m \in \mathbb{Z}} \sum_{n \in \mathbb{Z}} \langle x, \psi_{m,n} \rangle \cdot \psi_{m,n}(t)$$

where $\langle x, y \rangle$ is the scalar product $\int x \cdot \bar{y}$.

Appendix B

BACKPROPAGATION ALGORITHM

This section gives a brief overview of the backpropagation algorithm, commonly used to train neural networks, in conjunction with optimization techniques. Rojas provides a detailed description of the backpropagation algorithm [130]. The essential principle behind backpropagation is that at each iteration of the training process, the output regression error δ is computed and propagated backwards in order to compute new connection weights.

The backpropagation algorithm is an iterative one. Each iteration is composed of three main steps:

1. Feedforward computation
2. Backpropagation of the error
3. Weight updates

In the first step, the neural network is simulated with the current values of the connection weights. The output regression error δ is then calculated (as the difference between the output of the neural network and the actual target value in the training set).

In the second step, the error is propagated backwards so that the contribution of each neuron to the output error δ is estimated. This is done by reversing the directions of the connections in the neural network and by assigning the identity function as the transfer function of each neuron. As a result, if a neuron N_1 output is fed to two neurons N_2 and N_3 and if the error contributions of N_2 and N_3 are δ_2 and δ_3 respectively, then the error contribution for N_1 is defined as the weighted sum of δ_2 and δ_3 . This is illustrated in Figure 154.

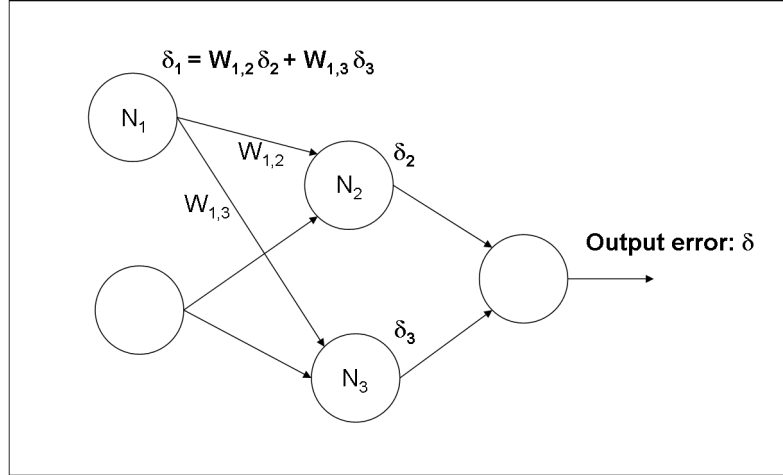


Figure 154: Error Backpropagation for Neuron N_1

In the third and final step of the training iteration, the contribution errors δ_i computed above are used to estimate the gradient of the error with respect to the connection weights w_i between the neurons. The connection weights w_i (or biases) are then updated accordingly. Rojas thoroughly describes this procedure [130].

Appendix C

MATLAB SOURCE CODE FOR WAVENET TRAINING

The Matlab source codes for training wavenets and processing the results in order to evaluate the goodness of fit are given in this section.

C.1 Training

```
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Wavenet_Training.m
%
% Training of the wavenet
% Optimization with fmincon
%
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
initMode = questdlg('Init mode or restart mode?','Starting mode','Init','Restart','Init');
%% Initial Options
%Number of Wavelons
nbWavelons= 10;
%Optimization parameters
maxIter=1000; %Max number of iterations
maxFunEvals=100000000; %Max number of function calls
%initMode='restart'; %'restart' or 'init'
if strcmp(initMode,'Init')==1
%% Input Training points
%Static input cases:
% xMat?
timeSamplesTrainingMat = [0:0.01:10];
tMat= timeSamplesTrainingMat;
tRes= tMat(2)-tMat(1); %constant time sampling
[casesInputMat, trainingDataMat]= getTrainingData(tMat);
[nbCases, nbDesignVar] = size(casesInputMat);
```

```

trainingSamplesMat = [tMat;trainingDataMat];
%[timesamples; signals_case1; signals_case2;...] for 1 test case
nbTrainingSamples = length(trainingSamplesMat(1,:));
ta= trainingSamplesMat(1,1);
tb= trainingSamplesMat(1, nbTrainingSamples);
%% Input Validation points
nbValidationCases = 10;
[casesValidationInputMat, validationDataMat]= getValidationData(tMat, nbValidationCases);
validationSamplesMat = [tMat;validationDataMat];
%[timesamples; signals_case1; signals_case2;...] for 1 test case
nbValidationSamples = length(validationSamplesMat(1,:));
end
%% Initialize the wavenet
% At the end of the wavenet initialization, we should have a lattice of
% alphas and betas for each static variable and for time t
% The outcome is also the parameter theta that will be the optimization
% input
if strcmp(initMode,'Init')==1
g0=mean(trainingSamplesMat(2,:));
wMat=10*ones(1,nbWavelons);
alphaMat=[];
betaMat=[];
%ta and tb
[alphaMat, betaMat] = initialize_x(ta, tb, nbWavelons);
for varNum=1:nbDesignVar
xa=min(casesInputMat(:,varNum));
xb=max(casesInputMat(:,varNum));
[alphaMatTemp, betaMatTemp] = initialize_x(xa, xb, nbWavelons);
alphaMat=[alphaMat, alphaMatTemp];
betaMat=[betaMat, betaMatTemp];
end
%theta is the parameter vector that will vary during optimization
theta0 = [g0, wMat, alphaMat, betaMat];
theta=theta0;
elseif strcmp(initMode,'Restart')==1
fprintf('\n%%%%%%%%%%%% RESTART FROM PREVIOUS OPTIMIZATION...
%%%%%%%%%%%%\n\n');
[FileName,PathName] = uigetfile('*.mat','Select the file of the optimization');
load(fullfile(PathName, FileName));
theta0=theta;
g0=mean(trainingSamplesMat(2,:));

```

```

tRes= tMat(2)-tMat(1); %constant time sampling
end
%% First Evaluations
xMat = casesInputMat(1,:);
fprintf('g0: %f', g0);
%calculate g_tilt (g_wavenet)
%g_wavenet = g0 + wavenetEval(timeSamplesTrainingMat,wMat, alphaMat, betaMat,nbTrainingSamples, nb-
Wavelons);

g_wavenet = wavenetEval(theta,tMat, xMat, nbTrainingSamples, nbWavelons);
%% Train the wavenet
tic;
%compute error (to be minimized)
%error = sqrt((g_wavenet- trainingSamplesMat(2,:)) * (g_wavenet- trainingSamplesMat(2,:))');
%threshold for error
epsilon= 1e-4;
%Time bounds for constrained optimization during training
lbTime= tMat(1)-10*tRes;
ubTime= tMat(length(tMat))+10*tRes;
%original = trainingSamplesMat(2, :);
%error = errorWavenet(theta,tMat, xMat, nbTrainingSamples, nbWavelons, trainingSamplesMat);
nbParamTheta = length (theta);
%gradError = grad (@errorWavenet, 0.00001, nbParamTheta, theta,tMat, xMat, nbTrainingSamples, nbWavelons,
original) ;
% Call errorWavenet(theta,tMat, nbSamples, nbWavelons, DataMat)
objectiveFun= @(thetaArg) errorWavenet (thetaArg,tMat, casesInputMat, nbTrainingSamples, nbWavelons, train-
ingDataMat);
% errorWavenet(theta,tMat, nbSamples, nbWavelons, DataMat)
fprintf('\nInitial point');
initValue = objectiveFun(theta);
fprintf('\nInitial point evaluation done');
LB = [];
UB= [];
fprintf('\ninitial error value: %f', initValue);
LB=-Inf*ones(1, length(theta));
UB=-LB;
LB(1, nbWavelons+2: 2*nbWavelons+1)= lbTime*ones(1, nbWavelons);
UB(1, nbWavelons+2: 2*nbWavelons+1)= ubTime*ones(1, nbWavelons);
options = optimset('Display','iter','MaxIter',maxIter, 'MaxFunEvals', maxFunEvals);
[X, FVAL,exitflag,output]= fmincon(objectiveFun, theta, [], [], [], [], LB, UB, [], options);
%[X, FVAL, history] = runfmincon(theta, LB, UB, tMat, casesInputMat, nbTrainingSamples, nbWavelons, train-
ingDataMat);

```

```

tElapsed =toc
theta=X;
%% Post Processing
Wavenet_PostProcessing;
fprintf('\n-----\n\n END Wavenet_Training\n\n-----');

```

C.2 *Post-Processing*

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% Post Processing of the Wavenet Training
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Get Predicted Values
Predicted_Training=zeros(nbCases, nbTrainingSamples);
Predicted_Validation=zeros(nbValidationCases, nbValidationSamples);
for i=1:nbCases
    Predicted_Training(i, :) = wavenetEval(theta,tMat, casesInputMat(i,:), nbTrainingSamples, nbWavelons);
end
for i=1:nbValidationCases
    Predicted_Validation(i, :) = wavenetEval(theta,tMat, casesValidationInputMat(i,:), ...
        nbValidationSamples, nbWavelons);
end
%% Calculate R-Square
R_Square=1-Calc_SSE(trainingDataMat,Predicted_Training)/Calc_SST(trainingDataMat);
R_Square_Validation=1-Calc_SSE([validationDataMat],[Predicted_Validation])/Calc_SST([validationDataMat]);
%% First Plots
%% Plot Actual vs Predicted
xmin=min(min(min(Predicted_Training), min(trainingDataMat)));
xmax=max(max(max(Predicted_Training), max(trainingDataMat)));
handle1=figure('name', 'Actual vs. Predicted', 'Visible', 'on');
%figure
hold on
title('Actual vs. Predicted');
XActVPred=[xmin, xmax];
plot(XActVPred, XActVPred, 'k')
plot(Predicted_Training, trainingDataMat, 'r. ');
handle2=figure('name', 'Actual vs. Predicted (with Validation Points)', 'Visible', 'on');
%figure
hold on
title('Actual vs. Predicted (with Validation Points)');
XActVPred=[xmin, xmax];
plot(XActVPred, XActVPred, 'k')

```

```

plot(Predicted__Training, trainingDataMat, 'r. ');
plot(Predicted__Validation, validationDataMat, 'b. ');
get(gca, 'Position')
annotation('textbox', get(gca, 'Position') + [0.02, 0.66, -0.52, -0.68], 'String', ...
str2mat(['R^2_{training}= ', num2str(R_Square)], ['R^2_{validation}= ', num2str(R_Square_Validation)]), ...
'FitBoxToText', 'on');
%% Plot Signals and Approximations
for i=1:nbCases
figure
plot(tMat, trainingDataMat(i,:))
hold
plot(tMat, Predicted__Training(i, :), 'r.')
legend('original signal', 'approximation');
title(['Case #' num2str(i)]);
end
%zoom out
tMat2=[-100:0.1:100];
g_wavenet2 = wavenetEval(theta, tMat2, casesInputMat(1,:), length(tMat2), nbWavelons);
%figure
handle3=figure('name', 'Zoom Out', 'Visible', 'on');
plot(tMat, trainingDataMat(1,:))
hold
plot(tMat2, g_wavenet2, 'r.')
legend('approximation');
%% Plot Residual vs. Predicted
handle8=figure('name', 'residualVPredicted', 'Visible', 'on');
plot(Predicted__Training, trainingDataMat-Predicted__Training, 'r. ');
%annotation('textbox', [x y w h])
hold
plot(Predicted__Validation, validationDataMat-Predicted__Validation, 'b. ');
plot([xmin, xmax], [0,0]);
title('Residual vs. Predicted');
xlabel('Predicted Value');
ylabel('Residual Error');
%% Plot Residual vs. Time
handle9=figure('name', 'residualVTime', 'Visible', 'on');
hold
ResidualTrainingMat=Predicted__Training-trainingDataMat;
ResidualValidationMat=Predicted__Validation-validationDataMat;
for i=1:length(Predicted__Training(:,1))
plot(tMat, ResidualTrainingMat(i,:), 'r. ');

```

```

end
for i=1:length(Predicted_Validation(:,1))
plot(tMat,ResidualValidationMat(i,:), 'b. ');
end
title('Residual vs. Time');
xlabel('Time');
ylabel('Residual');
%% Plot Distribution Relative Error
RelativeErrorTraining=100*ResidualTrainingMat./trainingDataMat;
handle10=figure('name', 'MFE_RelativeError', 'Visible', 'on');
[Distrib,xoutTraining] = hist(RelativeErrorTraining,1000);
a=sum(Distrib,2);
bar(xoutTraining,a);
%% Plot Distribution Residual (Absolute Error)
% Calculate means and standard deviations for training set and total set
ResidualTrainingMean=Calc_Mean(ResidualTrainingMat);
ResidualTrainingStdDev=Calc_STDDEV(ResidualTrainingMat);
ResidualTotalMean=Calc_Mean([ResidualTrainingMat;ResidualValidationMat]);
ResidualTotalStdDev=Calc_STDDEV([ResidualTrainingMat;ResidualValidationMat]);
handle11=figure('name', 'MFE_Residual', 'Visible', 'on');
[Distrib,xoutTraining] = hist(ResidualTrainingMat,30);
a=sum(Distrib,2);
bar(xoutTraining,a);
title('MFE: Distribution of the Residual on the Training Points (Absolute Error)');
xlabel('Residual');
ylabel('Count');
get(gca,'Position')
annotation('textbox',get(gca,'Position')+ [0.02,0.66,-0.54,-0.68], 'String',...
str2mat(['\mu_{MFE}= ', num2str(ResidualTrainingMean)],...
['\sigma_{MFE}= ', num2str(ResidualTrainingStdDev)]), 'FitBoxToText', 'on');
handle12=figure('name', 'MRE_Residual', 'Visible', 'on');
[Distrib,xoutTraining] = hist([ResidualTrainingMat;ResidualValidationMat],30);
a=sum(Distrib,2);
bar(xoutTraining,a);
title('MRE: Distribution of the Residual on all Points (Absolute Error)');
xlabel('Residual');
ylabel('Count');
annotation('textbox',get(gca,'Position')+ [0.02,0.66,-0.54,-0.68], 'String',...
str2mat(['\mu_{MRE}= ', num2str(ResidualTotalMean)],...
['\sigma_{MRE}= ', num2str(ResidualTotalStdDev)]), 'FitBoxToText', 'on');
%% Plot Original Signals and their Approximations

```



```

%Original function
%figure;
handle4=figure('name', 'Original Functions', 'Visible', 'on');
title('Original Functions');
hold on;
for i=1:nbCases
plot(tMat, trainingDataMat(i,:))
end
%Approximation
%figure;
handle5=figure('name', 'Approximations', 'Visible', 'on');
title('Approximations');
hold on;
for i=1:nbCases
plot(tMat, Predicted_Training(i, :), 'r')
end
%Original validation function
%figure;
handle6=figure('name', 'Original Validation Functions', 'Visible', 'on');
title('Original Validation Functions');
hold on;
for i=1:nbValidationCases
plot(tMat, validationDataMat(i,:))
end
%Approximated validation function
%figure;
handle7=figure('name', 'Approximated Validation Functions', 'Visible', 'on');
title('Approximated Validation Functions');
hold on;
for i=1:nbValidationCases
plot(tMat, Predicted_Validation(i,:), 'r')
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% SAVE
button = questdlg('Do you want to save the results?', 'Save Results');
if strcmp(button, 'Yes')
dateSerial= now;
direcSave = uigetdir('C:\Work\Thesis\Matlab Files','Select directory for figures');
filename= fullfile(direcSave, 'Results.mat') ;
save(filename, 'theta', 'tMat', 'timeSamplesTrainingMat', 'casesInputMat', 'trainingDataMat', ...
'casesValidationInputMat', 'trainingSamplesMat', 'nbCases', 'nbValidationCases',...

```

```

'nbTrainingSamples', 'nbValidationSamples', 'validationDataMat','nbWavelons', 'theta0', ...
'exitflag','output', 'maxIter', 'maxFunEvals', 'LB', 'UB','tElapsed', 'dateSerial') ;
%
type='.png';
for i=1:12
    handle=eval(strcat('handle',int2str(i)));
    saveas(handle, fullfile(direcSave, strcat(int2str(i), '-').get(handle, 'Name'), type)) );
end
end
end

```

C.3 Wavenet Output Evaluation and Definition of the Mother Wavelet

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COMPUTE WAVENET OUTPUT AND DEFINITION OF MOTHER WAVELET
%-----
% COMPUTE WAVENET OUTPUT
%-----

function wavenetResult= wavenetEval(theta,tMat, xMat, nbSamples, nbWavelons)
%wavenetResult is a vector: behavior of the wavelet approximation FOR ONE
%CASE
%alpha for translation
%beta for dilation
%dim alphaMat is nbWavelons
%dim betaMat is nbWavelons
%dim wMat is nbWavelons
%dim tMat is nbSamples
%dim wavenetResult is nbSamples
%nbSamples = length(tMat);
%nbWavelons = length(wMat);
nbVar=length(xMat)+1;
wMat=theta(2:nbWavelons+1);
fprintf('\nindex %d %d',nbWavelons+2,(nbVar+1)*nbWavelons+1) ;
fprintf('\nnbVar %d ',nbVar) ;
alphaVect= theta(nbWavelons+2:(nbVar+1)*nbWavelons+1);
fprintf('\nindex %d %d',(nbVar+1)*nbWavelons+2,(2*nbVar+1)*nbWavelons+1) ;
betaVect= theta((nbVar+1)*nbWavelons+2:(2*nbVar+1)*nbWavelons+1);
alphaMat=zeros(nbWavelons, nbVar);
betaMat=zeros(nbWavelons, nbVar);
for i=1:nbWavelons
    alphaMat(i,:)=alphaVect((i-1)*nbVar+1: i*nbVar);

```

```

betaMat(i,:)=betaVect((i-1)*nbVar+1: i*nbVar);
end

alpha_t_MatLarge= alphaMat(:,1) * ones(1,nbSamples);
beta_t_MatLarge= betaMat(:,1) *ones(1, nbSamples);
tMatLarge = ones(nbWavelons, 1) * tMat;
%tempMatrix= (tMatLarge-alphaMatLarge)./betaMatLarge;
wMatCompound= waveletTensorCompound(wMat, xMat, alphaMat, betaMat, nbWavelons);
wavenetResult = theta(1)+...
wMatCompound * wavedefMother((tMatLarge-alpha_t_MatLarge)./beta_t_MatLarge);
%-----
%-----
% DEFINITION OF MOTHER WAVELET
%-----
function psi = wavedefMother(t)
psi = -t .* exp(-0.5 * t.^2);
%-----
%-----
% MULTI DIMENSIONAL TENSOR WAVELET
%-----
function wMatCompound= waveletTensorCompound(wMat, xMat, alphaMat, betaMat, nbWavelons)
wMatCompound= zeros(1,nbWavelons);
numberStaticVar=length(xMat);
% for i_t=1:length(tMat)
% waveletResult(i_t)=wavedefMother((x-alpha)/beta)
% end
%instant t=tk=tMat(k)
for i=1:nbWavelons
%compute
prod_i=1;
for j=1:numberStaticVar
prod_i=prod_i * wavedefMother((xMat(j)-alphaMat(i,j+1))/betaMat(i,j+1));
end
wMatCompound(i) = wMat(i)*prod_i;
%sum=sum+wMat(i) * wavedefMother((t-alphaMat(i,1))/betaMat(i,1)) * prod_i;
end
%-----

```

Appendix D

JSL SCRIPT FOR THE GENERATION OF VISTRE

```
currentDir=Pick Directory("Select Directory");
    metaTable=CurrentDataTable();
    NumCases= NRows(metaTable);
    SignalRowsSelected={};
    SignalRowsSelectedStates={};
    metaTableRowStates={};
    ListSignalTables={};
    ListCasesSelected={};
    ListSignalsSelected={};
    windowsRef=Associative Array();
    windowsRef1=Associative Array();
    windowsRef2=Associative Array();
    windowsRef3=Associative Array();
    windowsRef4=Associative Array();
    cstrTable={};
    ListConstraintTables={};
    tMat={};
    cstrName="";
    tableName="";
    istart=0;
    iend=0;
    nSamples=0;
    ConstraintTypeMat={"Min", "Max"};
    ///-----
    // SELECT STATIC PARAMETERS AND DYNAMIC SIGNALS
    ///-----
    clusterDlg = NewWindow("Select Parameters, Responses and Signals",
    BorderBox(left(3),top(2),
    VListBox(
    HlistBox(
    VListBox(
```

```

PanelBox("Select Columns",
colListData=ColListBox(All,width(130),nLines(min(ncol(metaTable),10)))
)
),
PanelBox("Select",
LineupBox(NCol(2),Spacing(3),
ButtonBox("Design Parameters and Static Responses",
colListStat<<Append(colListData<<GetSelected);
),
colListStat= ColListBox(width(230, nLines(5), Numeric)),
ButtonBox("Time-Domain Signals",
colListY<<Append(colListData<<GetSelected)
),
colListY = ColListBox(width(230),nLines(5),character)
)
),
PanelBox("Action",
LineupBox(NCol(1),
ButtonBox("OK",
ListSignals=eval(colListY<<GetItems);
ListStaticParam=eval(colListStat<<GetItems);
if (NItems(ListSignals)==0 & NItems(ListStaticParam)==0, print("No column selected"),
Action;clusterDlg<<CloseWindow
)
),
ButtonBox("Cancel",clusterDlg<<CloseWindow),
TextBox(" "),
ButtonBox("Remove",
colListY<<RemoveSelected;
colListStat<<RemoveSelected;
),
//ButtonBox("Recall",notImplemented),
ButtonBox("Help",notImplemented))
)
)
)
);
///-----
// CREATE NEW DATA TABLE: AGREGATION OF ALL CASES FOR A SIGNAL
//-----

```

```

Action=expr(
For(SignalNum=1, SignalNum<=NItems(ListSignals),SignalNum++,
SignalName=ListSignals[SignalNum]; //SignalName is the name of the column (string)
Print("Processing Signal " ||SignalName);
//SignalFileInput = :Voltage<<get values;
SignalFileInput=Column(metaTable, SignalName)<<get values;
signalFile=Open(currentDir || signalFileInput[1]);
TimeColumn = :t<<get values;
nSamples=NRows(TimeColumn);
NumRows= NRows(SignalFile);
dt = New Table( SignalName||"Aggregate.jmp" );
dt<<Minimize Window;
dt<<NewColumn("Role",Character);
dt<<NewColumn("Case",Numeric);
dt<<NewColumn("t",Numeric);
dt<<NewColumn("Response",Numeric);
dt<<NewColumn("Selected_State", Numeric);
dt<<Begin Data Update;
For( caseNumber = 1, caseNumber <= NumCases, caseNumber++,
ResponseColumn= Column(SignalFile, caseNumber+1)<<get values;
For( i = 1, i <= NumRows, i++,
dt<< Add Rows({:Role="case", :Case=caseNumber, :t=TimeColumn[i],
:Response= ResponseColumn[i], :Selected_State =0});
//dt:Column(4)[i]=ResponseColumn[i];
);
);
Column(dt, "Response")<<Set Name(SignalName);
Close(signalFile);
dt<<End Data Update;
ListSignalTables[SignalNum]=dt;
cstrTable=New Table(SignalName||"Constraints");
cstrTable<<NewColumn("Name",Character);
//cstrTable<<NewColumn("ID", Numeric);
cstrTable<<NewColumn("Type", Character);
cstrTable<<NewColumn("Activity", Character);
cstrTable<<NewColumn("istart", Numeric);
cstrTable<<NewColumn("iend", Numeric);
cstrTable<<new column("Constraint Row States", row state);
cstrTable<<Minimize Window;
ListConstraintTables[SignalNum]=cstrTable;
); // End For

```

```

//-----
// CREATE VISUALIZATION WINDOWS
//-----

//Show(SignalName);

//Create Associative array: to each visualization window name is associated the reference of the table
mapWin2TableRef = Associative Array();

StaticWindow= New Window ("Visualization: Static Parameters",
mainBoxRef=OutlineBox("Transient Visualization Environment",
hboxRef=hlistbox(
OutlineBox("Static Parameters",
PanelBox("",
gb=metaTable<<Scatterplot Matrix( Y( eval(colListStat<<GetItems) ), Matrix Format( "Square" ) ),
hlistBox(
ButtonBox("UpdateSignals", UpdateSignals),
ButtonBox("Reset All", ResetAll),
ButtonBox("Close All", Close_All),
ButtonBox("TestWindowsRef", TestWindowsRef)
),
),
)
)
);

metaTable<<Select All Rows;
metaTable<<Colors (37);
metaTable<< Clear Select;

For(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
//Work with Associative Arrays
b2={};
dt={};
dt=ListSignalTables[SignalNum];
SignalName=ListSignals[SignalNum];
DynWindow=New Window("Visualization: Behavior of Signal "||SignalName,
b2=OutlineBox("Dynamic Signals",
Signal1=OutlineBox(SignalName,
panel1=PanelBox("",
overlay=dt<<Overlay Plot(
X( Column(dt,"t" )),
Y( Column(dt,SignalName) ),
Separate Axes( 1 ),
SendToReport( Dispatch( {}, "Overlay Plot", FrameBox, Frame Size( 428, 345 ) ) )

```

```

),
hlist box(
  PanelBox("Signal Selection",
    LineUpBox(NCol(1),
      ButtonBox("Highlight Signals", dt=CurrentDataTable();HighlightSignals),
      ButtonBox("Add to Selection", dt=CurrentDataTable();AddSignalToSelection),
      ButtonBox("Reset All", ResetAll),
      ButtonBox("Clear States", dt=CurrentDataTable();Clear_Row_States_Dyn),
      ButtonBox("Clear Selection",
        //overlay<<Data Table Window;
        //dt=Current Data Table ();
        SignalRef=mapWin2TableRef[SignalName];
        eval(FunClear_Selection_Dyn(dt));dt<<Select Where(1==0);),
        ButtonBox("Get TableName", GetDataTableName)
      )
    ),
    PanelBox("Constraints",
      LineUpBox(NCol(1),
        ButtonBox("Add Constraint",dt=CurrentDataTable();AddConstraint),
        ButtonBox("Remove Constraint", dt=CurrentDataTable();RemoveConstraint),
        ButtonBox("Highlight Constraint",dt=CurrentDataTable();HighlightConstraint),
        ButtonBox("Modify Constraint Properties",dt=CurrentDataTable();ModifyConstraint),
        ButtonBox("Redraw", dt=CurrentDataTable();Redraw);
      );
    ),
    PanelBox("Filtering",
      LineUpBox(NCol(1),
        ButtonBox("Highlight Undesirable Points", FilterSignals),
        ButtonBox("Highlight Desirable Signals", KeepSelection),
        //ButtonBox("Hide Selection", HideSelection)
      );
    ),
    PanelBox("Propagate",
      LineUpBox(NCol(1),
        ButtonBox("Update", UpdateStaticPoints),
        //ButtonBox("Propagate markers", dt=CurrentDataTable();PropagateMarkers)
      );
    )
  )
)
)
)
)
)
)

```



```

)
);
dt<< Select All Rows;
dt<<Colors (37);
dt<< Clear Select;
windowsRef1[dt<<GetName]=DynWindow;
windowsRef2[dt<<GetName]=dt;
windowsRef3[dt<<GetName]=SignalNum;
windowsRef4[dt<<GetName]=b2;
mapWin2TableRef[Signal1<<Get Title]=dt;
);
); //End of Action Script
//-----
// SCRIPTS ON THE DYNAMIC SIGNALS SIDE
//-----
Redraw=expr(
dt=CurrentDataTable();
//dt<<AddRows({:Role="constraint", :Case=1, :t=5, :Selected_State=0}); //FOR TEST PURPOSES ONLY
dtName=dt<<Get Name;
Show(dtName);
DynWindow=windowsRef1[dtName];
pos=DynWindow<<Get Window Position;
//windowsRef2[dt<<GetName]=dt;
SignalNum=windowsRef3[dtName];
//windowsRef4[dtName];
Show(SignalNum);
DynWindow<<Close Window;
//For(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
//Work with Associative Arrays
//b2={};
//dt={};
//dt=ListSignalTables[SignalNum];
SignalName=ListSignals[SignalNum];
Show(SignalName);
//Column(dt,SignalName)[NRows(dt)]=8;
DynWindow=New Window("NEWVisualization: Behavior of Signal "||SignalName,
b2=OutlineBox("Dynamic Signals",
Signal1=OutlineBox(SignalName,
panel1=PanelBox("",
overlay=dt<<Overlay Plot(
X( Column(dt,"t" )),

```

```

Y( Column(dt,SignalName) ),
Separate Axes( 1 ),
SendToReport( Dispatch( {}, "Overlay Plot", FrameBox, Frame Size( 428, 345 ) ) )
),
hlist box(
PanelBox("Signal Selection",
LineUpBox(NCol(1),
ButtonBox("Highlight Signals", dt=CurrentDataTable();HighlightSignals),
ButtonBox("Add to Selection", dt=CurrentDataTable();AddSignalToSelection),
ButtonBox("Reset All", ResetAll),
ButtonBox("Clear States", dt=CurrentDataTable();Clear_Row_States_Dyn),
ButtonBox("Clear Selection",
//overlay<<Data Table Window;
//dt=Current Data Table ();
SignalRef=mapWin2TableRef[SignalName];
eval(FunClear_Selection_Dyn(dt));dt<<Select Where(1==0);),
ButtonBox("Get TableName", GetDataTableName)
)
),
PanelBox("Constraints",
LineUpBox(NCol(1),
ButtonBox("Add Constraint",dt=CurrentDataTable();AddConstraint),
ButtonBox("Remove Constraint", dt=CurrentDataTable();RemoveConstraint),
ButtonBox("Highlight Constraint",dt=CurrentDataTable();HighlightConstraint),
ButtonBox("Modify Constraint Properties",dt=CurrentDataTable();ModifyConstraint),
ButtonBox("Redraw", dt=CurrentDataTable();Redraw);
);
),
PanelBox("Filtering",
LineUpBox(NCol(1),
ButtonBox("Highlight Undesirable Points", FilterSignals),
ButtonBox("Highlight Desirable Signals", HighlightDesirable),
//ButtonBox("Hide Selection", HideSelection)
);
),
PanelBox("Propagate",
LineUpBox(NCol(1),
ButtonBox("Update", UpdateStaticPoints),
//ButtonBox("Propagate markers", dt=CurrentDataTable();PropagateMarkers)
);
)

```

```

)
)
)
)
);

newPos=DynWindow<<Get Window Position;
x=pos[1];y=pos[2];
xnew=newPos[1]; ynew=newPos[2];
show(x); Show(y); Show(xnew); Show(ynew);
//DynWindow<<Move Window(x-xnew, y-ynew);
DynWindow<<Move Window(pos);
windowsRef1[dt<<GetName]=DynWindow;
windowsRef2[dt<<GetName]=dt;
windowsRef3[dt<<GetName]=SignalNum;
windowsRef4[dt<<GetName]=b2;
mapWin2TableRef[Signal1<<Get Title]=dt;
);

GetDataTableName=expr(
name=CurrentDataTable();
print(name<<Get Name);
name:Selected_State[1]=10;
Column(name, "Selected_State")[1]=20;
);

AddSignalToSelection=expr(//APPEND LISTCASESSELECTED
FindSelectedSignals;//Updates ListSignalsSelected
For(i=1, i<=NItems(ListSignalsSelected),i++,
if(!Contains(ListCasesSelected, ListSignalsSelected[i]),
ListCasesSelected[NItems(ListCasesSelected)+1]=ListSignalsSelected[i];
)
);

Show(ListCasesSelected);//TEST
);

PropagateMarkers=expr(
print("Propagate for "||(dt<<Get Name));
//SignalRowsSelected={};
//SignalRowsSelectedStates={};
GetRowStates;
for(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
dt=ListSignalTables[SignalNum];
Current DataTable(dt);
print("Propagating Row States for "||(dt<<GetName));

```

```

dt<<Begin Data Update;
//Show(NItems(SignalRowsSelectedStates));
//Show(NRows(SignalRowsSelected));
//Show(SignalRowsSelected);
for(j=1, j<=NItems(SignalRowsSelectedStates), j++,
rowstate(SignalRowsSelected[j])=SignalRowsSelectedStates[j];
);
dt<<End Data Update;
//dt<<Select Where(:Selected_State==1);
);
print("END PropagateMarkers");
);
GetRowStates=expr("//Find and "return" row states of selection
print("Propagate for "||(dt<<Get Name));
//dtSel=dt<<Next Selected;
//SignalRowsSelected={};
//SignalRowsSelectedStates={};
SignalRowsSelected=dt<<Get Selected Rows;
//Show(NRows(SignalRowsSelected));
SignalRowsSelectedStates={};
SignalNum=windowsRef3[dt<<GetName];
cstrTable=ListConstraintTables[SignalNum];
For(i=1, i<= NRows(SignalRowsSelected), i++,
if(Column(dt,"case")[SignalRowsSelected[i]]<=0,
//Revert Row State Constraint
cstrName=Column(dt,"Role")[SignalRowsSelected[i]];
cstrRow=cstrTable<<Get Rows Where(:Name==cstrName);
rowState(SignalRowsSelected[i])=Column(cstrTable, "Constraint Row States")[cstrRow[1]];
print("Reverting Constraint Row State"),
//ELSE
SignalRowsSelectedStates[i]=RowState(SignalRowsSelected[i]);
);
);
//show(SignalRowsSelectedStates);
print("END GetRowStates");
);
HighlightSignals=expr(
//CLEAR SELECTION
Clear_Selection_Dyn;
//nCases=0;
//DETERMINE LIST OF CASES SELECTED IN SIGNAL PLOT

```

```

FindSelectedSignals;
Print("DONE FIND SELECTED SIGNALS");
if(NItems(ListSignalsSelected)==0, print("No signal selected"),
//show(ListCases);
//SELECT ENTIRE SIGNALS
dt<<Begin Data Update;
For(i=1, i<=NItems(ListSignalsSelected),i++,
Show(ListSignalsSelected[i]);//TEST
currentCase=ListSignalsSelected[i];
// y=dt<<Get Rows Where(:Case==ListSignalsSelected[i]);
// Show(NRows(y));//TEST
// For(j=1, j<=NRows(y), j++,
// //Show(y[j]);//TEST
// //dt:Selected_State[y[j]]=1;
// //dt:Selected_State[1]=100;//TEST
// Column(dt, "Selected_State")[y[j]]=1;
// rowState(y[j])=color state(37);
// );
//y<<Values(repeat(9,1));
//dt<<ColorOf(row state(i)) = 3;
For(j=(currentCase-1)*nSamples+1, j<=currentCase*nSamples, j++,
Column(dt, "Selected_State")[j]=1;
rowState(j)=color state(0);
);
);
dt<<End Data Update;
dt<<Select Where(:Selected_State==1);
);
);
ResetAll=expr(
for(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
dt=ListSignalTables[SignalNum];
Current Data Table(dt);
Clear_Selection_Dyn;
Clear_Row_States_Dyn;
//dt<<Select Where(1==0);
//dt<< Select All Rows;
//dt<<Colors (37);
dt<< Clear Select;
);
Clear_Selection_Static;

```

```

Clear_Row_States_Static;
);
ResetAllDyn=expr(
for(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
dt=ListSignalTables[SignalNum];
Current DataTable(dt);
Clear_Selection_Dyn;
Clear_Row_States_Dyn;
dt<<Select Where(1==0);
);
//Clear_Selection_Static;
//Clear_Row_States_Static;
);
Clear_Row_States_Dyn=expr(
dt=CurrentDataTable();
Print("Clear Row States for "||(dt<<Get Name));
//dt<<Clear Row States;
ListSignalsSelected={};
listCaseRows=dt<<Get Rows Where (:Role=="case");
for(i=1, i<=NRows(listCaseRows), i++,
rowState(listCaseRows[i])=CombineStates(color state(37), marker state(0),hidden state(0), excluded state(0));
);
);
Clear_Selection_Dyn_one=expr(//Clears selection of one signal window
//dt=CurrentDataTable();
print("Clear Selection for "||(dt<<Get Name()));
y=dt<<Get Rows Where(:Selected_State==1);
For(j=1, j<=NRows(y), j++,
//dt:Selected_State[y[j]]=0;
Column(dt,"Selected_State")[y[j]]=0;
);
ListSignalsSelected={};
//dt<<Select Where(1==0);
);
Clear_Selection_Dyn=expr(
dt=CurrentDataTable();
print("Clear Selection for "||(dt<<Get Name()));
y=dt<<Get Rows Where(:Selected_State==1);
For(j=1, j<=NRows(y), j++,
//dt:Selected_State[y[j]]=0;
Column(dt,"Selected_State")[y[j]]=0;

```

```

);
ListSignalsSelected={};
//dt<<Select Where(1==0);
);
FunClear_Selection_Dyn=Function({dt}, Clear_Selection_Dyn);
Clear_Selection_Dyn_ALL=expr("//CLEAR SELECTION OF ALL SIGNAL WINDOWS
//dt=CurrentDataTable();
print(dt<< Get Name());
y=dt<<Get Rows Where(:Selected_State==1);
For(j=1, j<=NRows(y), j++,
//dt:Selected_State[y[j]]=0;
Column(dt,"Selected_State")[y[j]]=0;
);
ListSignalsSelected={};
//dt<<Select Where(1==0);
);
FindSelectedSignals=expr("//From selected points on graph, find the corresponding case numbers
print("FindSelectedSignals for "|| (dt<< Get Name()));
selectedDynPoints = dt<<Get Selected Rows;
nSelected=NRows(selectedDynPoints);
//ListCases={};
if(nSelected==0, print("No signal selected"),
nCases=NItems(ListSignalsSelected);
Show(nSelected);
For(i= 1, i<=nSelected, i++,
//Show(i);
//caseValue=dt:Case[selectedDynPoints[i]];
caseValue=Column(dt,"Case")[selectedDynPoints[i]];
//Show(caseValue);
if(caseValue<=0,selectedDynPoints[i]=0, //DESELECT CONSTRAINTS
if(!Contains(ListSignalsSelected, caseValue), ListSignalsSelected[nCases+1]=caseValue;nCases++);
//dt<<Select Where(:Case==caseValue));
);
);
Show(NItems(ListSignalsSelected));//TEST
);
);
SelectSignals=expr("// MIGHT BE OBSOLETE
//numRows=dt<<NRows();
//selectedDynPoints = dt<<Get Selected Rows;
//nSelected=NRows(selectedDynPoints);

```

```

dt=CurrentDataTable();
print(dt<< Get Name());
//CLEAR SELECTION
Clear_Selection_Dyn;
//nCases=0;
//DETERMINE LIST OF CASES SELECTED IN SIGNAL PLOT
FindSelectedSignals;
//Show(ListSignalsSelected);
if(NItems(ListSignalsSelected)==0, print("No signal selected"),
//show(ListCases);
//SELECT ENTIRE SIGNALS
dt<<Begin Data Update;
For(i=1, i<=NItems(ListSignalsSelected),i++,
Show(ListSignalsSelected[i]);//TEST
y=dt<<Get Rows Where(:Case==ListSignalsSelected[i]);
Show(NRows(y));//TEST
For(j=1, j<=NRows(y), j++,
//Show(y[j]);//TEST
//dt:Selected_State[y[j]]=1;
//dt:Selected_State[1]=100;//TEST
Column(dt, "Selected_State")[y[j]]=1;
);
//y<<Values(repeat(9,1));
//dt<<ColorOf(row state(i)) = 3;
);
dt<<End Data Update;
dt<<Select Where(:Selected_State==1);
);
);
//theList= selectedDynPoints<<(:Case<<get values);
//if (Contains(theList,2), b2<< Append(t1 = Text Box (2 is there!)));
UpdateStaticPoints=expr(
Clear_Selection_Dyn;
//DETERMINE LIST OF CASES SELECTED IN SIGNAL PLOT
//FindSelectedSignals;
//ASSIGN SELECTED SIGNALS TO STATIC CASES
//ListCasesSelected=ListSignalsSelected; Should already be done
nSelected=NItems(ListcasesSelected);
if(nSelected==0, print("No signal selected"),
PropagateMarkers;
row_state=SignalRowsSelectedStates[1];

```



```

CurrentDataTable(metaTable);
For(i=1, i<=nSelected,i++,
// y=metaTable<<Get Rows Where(:Case==ListCasesSelected[i]);
// For(j=1, j<=NRows(y), j++,
// metaTable:Selected_State[y[j]]=1;
// rowState(y[j])=row_state;
// );
metaTable:Selected_State[ListCasesSelected[i]]=1;
rowState(ListCasesSelected[i])=row_state;
//metaTable<<Select Where(:Selected_State==1);
);
BringWindowsToFront;
);
//HighlightSelectedSignals;
);
//-----
// SCRIPTS ON THE STATIC SCATTERPLOT SIDE
//-----
Clear_Row_States_Static=expr(
//dt=CurrentDataTable();
Print("Clear Row States for "||(metaTable<<Get Name));
metaTable<<Clear Row States;
metaTable<<Select All Rows;
metaTable<<Colors (37);
metaTable<<Clear Select;
);
Clear_Selection_Static=expr(
//metaTable<<Select Where(1==0);
y=metaTable<<Get Rows Where(:Selected_State==1);
For(j=1, j<=NRows(y), j++,
metaTable:Selected_State[y[j]]=0;
);
ListCasesSelected={};
);
UpdateSignals=expr(
selectedStatPoints = metaTable<<Get Selected Rows;
if(NRows(selectedStatPoints)==0, print("No Case Selected"),
ResetAllDyn;
Clear_Selection_Static;
metaTableRowStates={};
CurrentDataTable(metaTable);

```

```

For(i=1, i<=NRows(selectedStatPoints), i++,
metaTable:Selected_State[selectedStatPoints[i]]=1;
ListCasesSelected[i]=metaTable:Case[selectedStatPoints[i]];
metaTableRowStates[i]=rowState(selectedStatPoints[i]);
);
//metaTable<<Select Where(:Selected_State==1);
ListSignalsSelected=ListCasesSelected;
//SELECT ENTIRE SIGNALS
for(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
dt=ListSignalTables[SignalNum];
Current Data Table(dt);
print("Selecting signals for "||(dt<<GetName));
dt<<Begin Data Update;
For(i=1, i<=NItems(ListCasesSelected),i++,
//Show(ListCases[i]);
rowState_i=metaTableRowStates[i];
//y=dt<<Get Rows Where(:Case==ListCasesSelected[i]);
currentCase=ListCasesSelected[i];
//For(j=1, j<=NRows(y), j++,
// Column(dt, "Selected_State")[y[j]]=1;
// rowstate(y[j])=rowState_i;
//);
//y<<Values(repeat(9,1));
//dt<<ColorOf(row state(i)) = 3;
For(j=(currentCase-1)*nSamples+1, j<=currentCase*nSamples, j++,
Column(dt, "Selected_State")[j]=1;
rowState(j)=rowState_i;
);
);
dt<<End Data Update;
//dt<<Select Where(:Selected_State==1);
);
BringWindowsToFront;
);
);
UpdateSignals_OLD=expr(//Obsolete
selectedStatPoints = metaTable<<Get Selected Rows;
if(NRows(selectedStatPoints)==0, print("No Case Selected"),
ResetAll;
Clear_Selection_Static;
For(i=1, i<=NRows(selectedStatPoints), i++,

```

```

metaTable:Selected_State[selectedStatPoints[i]]=1;
ListCasesSelected[i]=metaTable:Case[selectedStatPoints[i]];
);
metaTable<<Select Where(:Selected_State==1);
//CLEAR SELECTION
//Clear_Selection_Dyn;
//dt<<Select Where(1==0);
//nCases=0;
ListSignalsSelected=ListCasesSelected;
//Show(ListSignalsSelected);
//show(ListCases);
//SELECT ENTIRE SIGNALS
for(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
dt=ListSignalTables[SignalNum];
Current Data Table(dt);
print("Selecting signals for "||(dt<<GetName));
dt<<Begin Data Update;
For(i=1, i<=NItems(ListCasesSelected),i++,
//Show(ListCases[i]);
y=dt<<Get Rows Where(:Case==ListCasesSelected[i]);
For(j=1, j<=NRows(y), j++,
Column(dt, "Selected_State")[y[j]]=1;
);
//y<<Values(repeat(9,1));
//dt<<ColorOf(row state(i)) = 3;
);
dt<<End Data Update;
dt<<Select Where(:Selected_State==1);
);
);
);
HighlightSelectedSignals=expr(
for(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
dt=ListSignalTables[SignalNum];
Current Data Table(dt);
print("Selecting signals for "||(dt<<GetName));
dt<<Begin Data Update;
For(i=1, i<=NItems(ListCasesSelected),i++,
//Show(ListCases[i]);
y=dt<<Get Rows Where(:Case==ListCasesSelected[i]);
For(j=1, j<=NRows(y), j++,

```

```

Column(dt, "Selected_State")[y[j]]=1;
);
//y<<Values(repeat(9,1));
//dt<<ColorOf(row state(i)) = 3;
);
dt<<End Data Update;
dt<<Select Where(:Selected_State==1);
);
);
Close_All=expr(
for(SignalNum=1, SignalNum<=NItems(List Signals), SignalNum++,
//Show(SignalNum);
tableRef=ListSignalTables[SignalNum];
print("Will close table "||(tableRef<<GetName));
//CurrentDataTable(tableRef);
boxRef=(windowsRef4[tableRef<<GetName]);
Show(boxRef);
title=boxRef<<GetTitle;
print("Closing Window " || title);
boxRef<<Close Window;
Close(tableRef, nosave);
Close(ListConstraintTables[signalNum], nosave);
//windowsRef[dt<<GetName]={DynWindow, dt, SignalNum, b2};//
//mapWin2TableRef[Signal1<<Get Title]=dt;
);
);
//-----
// CONSTRAINTS SCRIPTS
//-----
AddConstraint=expr(
SignalNum=windowsRef3[dt<<GetName];
SignalName=ListSignals[SignalNum];
newOrExisting=NewWindow(" Add a constraint",
BorderBox(left(3),top(2),
hlistBox(
PanelBox("",
vlistBox(
hlistBox(
tb1=TextBox("Name: "),/<<Bullet Point(1);
teb=TextEditBox("", <<Script(Print(teb<<Get Text)));
),

```

```

vlistBox(
tb2=TextBox("Define a new constraint or load a predefined one:"),
hlistBox(
  ButtonBox("New Constraint", cstrName=teb<<Get Text;newConstraint=1;
newOrExisting<<CloseWindow;AddNewConstraint),
  ButtonBox("Load Constraint", cstrName=teb<<Get Text;newConstraint=0;
newOrExisting<<CloseWindow;LoadConstraint)
);
),
);
),
ButtonBox("Cancel", Print("Canceling"); newOrExisting<<CloseWindow;));
);
);
);
tb1<<Bullet Point(1);
tb2<<Bullet Point(1);
);
AddNewConstraint=expr(
cTable="Not Defined";
cstrDlg=NewWindow("Add Constraint",
  BorderBox(left(3),top(2),
//dt = New Table( SignalName||"Aggregate.jmp" );
hlistBox(
vlistBox(
hlistBox(
tb=TextBox("Name: "),//<<Bullet Point(1);
teb=TextEditBox(cstrName, <<Script(Print(teb<<Get Text)));
),
PanelBox("Type",
  ConstraintType=RadioBox({"Min", "Max"});
),
//a=edit number(1);
//Show(a);
),
TextBox(" "),
vlistBox(
  ButtonBox("Make Constraint Table", cstrName=teb<<Get Text;MakeConstraintTable),
  ButtonBox("OK",
print("OK!!!");
if(cTable=="Not Defined", print("Error: Constraint not defined"),

```

```

tMat=Column(cTable,"t")<<Get As Matrix;
ResponseMat=Column(cTable,SignalName)<<Get AsMatrix;
//ResponseMat[NItems(Column(cTable,SignalName))];
//Show(ResponseMat);
cTable<<Minimize Window;
);
cstrDlg<<Close Window;
AppendConstraintToAggregate;
Redraw;
),
ButtonBox("Cancel", print("Cancelling!!!");cstrDlg<<Close Window;),
)
)
),
);
tb<<Bullet Point(1);
cstrDlg<<Size Window(300, 250);
);
AppendConstraintToAggregate=expr(
//cstrID=1;
Print("Calling AppendConstraintToAggregate");//For Testing
Show(SignalNum);//For Testing
dt<<New Column(cstrName, Numeric);
cstrTable=ListConstraintTables[SignalNum];
cstrTable<<AddRows({:Name=cstrName, :Type=ConstraintTypeMat[typ],
:Activity="Active", :Constraint Row States=combine states(color state(19), marker state(0))});
Show(typ);
//if(typ=="Min", cstrTypeNum=-2, type=="Max", cstrTypeNum=-1, cstrTypeNum=0);
cstrTypeNum=typ-3;
Show(tMat);
Show(ResponseMat);
Show(dt<<GetName);
//Find TimeColumn
SignalFileInput=Column(metaTable, SignalName)<<get values;
signalFile=Open(currentDir || signalFileInput[1]);
TimeColumn = :t<<get values;
Close(SignalFile);
//Show(NRows(TimeColumn));
//Find the index at which the constraint starts applying to dt
//by scanning TimeColumn until time is bigger that first element of constraint time column
For(istart=1, (istart<=NRows(TimeColumn)& TimeColumn[istart]<tMat[1]), istart++,

```

```

//Do nothing
istart;
);
//Print("Done with istart");//For Testing
//Show(istart);//For Testing
//Find the index at which the constraint stops applying to dt
//by scanning TimeColumn from istart until time is bigger than last element of constraint time column
For(iend=istart, (iend<=NRows(TimeColumn) & TimeColumn[iend]<=tMat[NRows(tMat)]),
iend++, iend);
//Print("Done with iend");//For Testing
//Show(iend); //For Testing
Column(cstrTable,"istart")[NRows(cstrTable)]=istart;
Column(cstrTable,"iend")[NRows(cstrTable)]=iend;
nSamples=NRows(TimeColumn);
temp=newTable("tempTable", invisible);
dt<<Begin Data Update;
temp<<Begin Data Update;
temp<<NewColumn("Role",Character);
temp<<NewColumn("Case",Numeric);
temp<<NewColumn("t",Numeric);
temp<<NewColumn(SignalName,Numeric);
temp<<NewColumn("Selected_State", Numeric);
//cursor=1;
//Print("Starting populating Temp");//For Testing
//Show(istart);
For(j=istart, j<=iend-1, j++,
temp<<AddRows({:Role=cstrName, :Case=cstrTypeNum, :t=TimeColumn[j], :Selected_State=0});
yInterp=Interpolate(TimeColumn[j], tMat, ResponseMat);
Column(temp,SignalName)[j-istart+1]=yInterp;
rowstate(j-istart+1)=combine states(color state(19), marker state(0));
For(k=1, k<=NumCases,k++,
Column(dt,cstrName)[(k-1)*nSamples+j]=yInterp;
);
);
temp<<End Data Update;
dt << Concatenate(temp, Append to first table);
dt<<End Data Update;
Close(temp, No Save);
);
MakeConstraintTable=expr(
//cstrName=teb<<GetText;

```

```

cTable=NewTable(cstrName);
cTable<<NewColumn("Constraint Type",Character);
cTable<<NewColumn("t",Numeric);
//SignalNum=windowsRef3[dt<<GetName];
//SignalName=ListSignals[SignalNum];
Show(SignalName);
cTable<<NewColumn(SignalName, Numeric);
typ=ConstraintType<<Get;
//Show(typ);
//Show(ConstraintTypeMat[typ]);
cTable<<AddRows({:Constraint Type=ConstraintTypeMat[typ]});
//Column(cTable,"Constraint Type")[1]="Alors????";
);
LoadConstraint=expr(
//LOAD CONSTRAINT PREVIOUSLY SAVED AS JMP TABLE
cTableFileName=Pick File("Select a Data Table", currentDir, {"JMP Files\jmp"});
cTable=Open(cTableFileName);
colNames=cTable<<Get Column Names(String);
if(!Contains(colNames,"Constraint Type")|!Contains(colNames,"t")|!Contains(colNames,SignalName),
//TO DO: CHECK THAT FORMAT IS CORRECT
print("Error, not a constraint");
h=Dialog(Title("Error"),"Not a constraint table!");
Close(cTable),
//ELSE
print("Good constraint");
//Get Attributes
typString=Column(cTable, "Constraint Type")[1];
if(typString=="Min", typ=1, typString=="Max", typ=2);
tMat=Column(cTable,"t")<<get values;
ResponseMat=Column(cTable,SignalName)<<get values;
if(cstrName=="", cstrName=cTable<<Get Name);
Close(cTable);
AppendConstraintToAggregate;
Redraw;
);
);
RemoveConstraint=expr(
SignalNum=windowsRef3[dt<<GetName];
DynWindow=windowsRef1[dt<<GetName];
SignalName=ListSignals[SignalNum];
cstrTable=ListConstraintTables[SignalNum];

```



```

selectWindow=New Window("Select constraint to be deleted",
BorderBox(left(3),top(2),
cstrNameList=Column(cstrTable,"Name")<<Get values;
Show(cstrNameList);
hlistBox(
PanelBox("Select constraint to be deleted",
radioB=RadioBox(cstrNameList);
),
PanelBox("Action",
ButtonBox("Delete",
cstrNameNum=radioB<<Get;
cstrName=cstrNameList[cstrNameNum];
print("Delete "||cstrName);//TEST
DeleteSelectedConstraint;
selectWindow<<Close Window;
CurrentDataTable(dt);
Redraw;
),
ButtonBox("Cancel", print("Cancel");selectWindow<<Close Window;);
);
);
)
);
);
DeleteSelectedConstraint=expr(
//Name: cstrName
print("Calling DeleteSelectedConstraint For "||cstrName);//For Testing
//dt=windowsRef2[SignalName];
ListSignalTables[SignalNum]=dt;
dt<<Delete Columns(cstrName);
cstrRows=dt<<Select Where (:Role==cstrName);
dt<<Delete Rows(cstrRows);
cstrRows=cstrTable<<Select Where (:Name==cstrName);
cstrTable<<Delete Rows(cstrRows);
);
HighlightConstraint=expr(
//get Min/Max, Active, Name, Color, Marker
SignalNum=windowsRef3[dt<<GetName];
DynWindow=windowsRef1[dt<<GetName];
SignalName=ListSignals[SignalNum];
cstrTable=ListConstraintTables[SignalNum];

```

```

selectWindow=New Window("Select constraint to be highlighted",
BorderBox(left(3),top(2),
cstrNameList=Column(cstrTable,"Name")<<Get values;
Show(cstrNameList);
hlistBox(
PanelBox("Select constraint to be highlighted:",
radioB=RadioBox(cstrNameList);
),
PanelBox("Action",
ButtonBox("Highlight",
cstrNameNum=radioB<<Get;
cstrName=cstrNameList[cstrNameNum];
//print("Delete "||cstrName);//TEST
HighlightSelectedConstraint;
selectWindow<<Close Window;
),
ButtonBox("Cancel", print("Cancel");selectWindow<<Close Window;);
);
);
);
);
);
);
HighlightSelectedConstraint=expr(
//cstrNameNum
dt<<Select Where(:Role==cstrName);
);
ModifyConstraint=expr(
SignalNum=windowsRef3[dt<<GetName];
DynWindow=windowsRef1[dt<<GetName];
SignalName=ListSignals[SignalNum];
cstrTable=ListConstraintTables[SignalNum];
selectWindow=New Window("Select constraint to be modified",
BorderBox(left(3),top(2),
cstrNameList=Column(cstrTable,"Name")<<Get values;
Show(cstrNameList);
hlistBox(
PanelBox("Select constraint to be modified:",
radioB=RadioBox(cstrNameList);
),
PanelBox("Action",
ButtonBox("Modify",

```

```

cstrNameNum=radioB.<<Get;
cstrName=cstrNameList[cstrNameNum];
print("Delete "||cstrName);//TEST
ModifySelectedConstraint;
selectWindow.<<Close Window;
),
ButtonBox("Cancel", print("Cancel");selectWindow.<<Close Window;);
);
);
);
);
);
);
ModifySelectedConstraint=expr(
//Get Min/Max Active Name
activityOld=Column(cstrTable, "Activity")[cstrNameNum];
activityIndex=0;
if(activityOld=="Active", activityIndex=1, activityIndex=2);
typeOld=Column(cstrTable, "Type")[cstrNameNum];
typeIndex=0;
if(typeOld=="Min", typeIndex=1, typeIndex=2);
cstrNameOld=cstrName;
cstrPropertyWindow=New Window("Constraint Properties for "|| cstrName,
BorderBox(left(3),top(2),
hlistBox(
vlistBox(
hlistBox(
tb=TextBox("Name: "),//<<Bullet Point(1);
teb=TextEditBox(cstrName, <<Script(Print(teb.<<Get Text)));
),
hlistBox(
PanelBox("Type",
ConstraintType=RadioBox({"Min", "Max"},<<Set(typeIndex));
),
PanelBox("Activity",
ConstraintActivity=RadioBox({"Active", "Inactive"},<<Set(activityIndex));
),
),
//a=edit number(1);
//Show(a);
),
TextBox(" "),

```

```

vlistBox(
//ButtonBox("Make Constraint Table", cstrName=teb<<Get Text;MakeConstraintTable),
ButtonBox("OK", //Get new Properties and modify appropriately
print("OK!!!");
cstrNameNew=teb<<Get Text;
typeNewIndex=ConstraintType<<Get;
typeNew={"Min", "Max"}[typeNewIndex];
activityNewIndex=ConstraintActivity<<Get;
activityNew={"Active", "Inactive"}[activityNewIndex];
if(cstrNameNew!=cstrNameOld, ModifyConstraintName);
if(typeNew!=typeOld, ModifyConstraintType);
if(activityNew!=activityOld, ModifyConstraintActivity);
cstrPropertyWindow<<Close Window;
CurrentDataTable(dt);
Redraw;
),
ButtonBox("Cancel", print("Cancelling!!!");cstrPropertyWindow<<Close Window;),
)
)
);
);
);
ModifyConstraintName=expr(
//New name: cstrNameNew
//Old name: cstrNameOld
//cstrNameNum
Column(cstrTable, "Name")[cstrNameNum]=cstrNameNew;
rowsCstr=dt<<Get Rows Where(:Role==cstrNameOld);
for(i=1, i<=NRows(rowsCstr), i++,
Column(dt, "Role")[rowsCstr[i]]=cstrNameNew;
);
Column(dt,cstrNameOld)<<Set Name(cstrNameNew);
);
ModifyConstraintType=expr(
//New Type: typeNew
//Old Type: typeOld
//cstrNameNum
Column(cstrTable, "Type")[cstrNameNum]=typeNew;
rowsCstr=dt<<Get Rows Where(:Role==cstrNameOld);
if(typeNew=="Min", typ=-2, typ=-1);
for(i=1, i<=NRows(rowsCstr), i++,

```

```

Column(dt, "Case")[rowsCstr[i]]=typ;
);
);
ModifyConstraintActivity=expr(
//New Type: typeNew
//Old Type: typeOld
//cstrNameNum
Column(cstrTable, "Activity")[cstrNameNum]=activityNew;
);
//-----
// SCRIPTS FOR FILTERING
//-----
FilterSignals=expr(//SELECT POINTS THAT VIOLATE CONSTRAINTS
dt=CurrentDataTable();
tableName=dt<<getName;
ResetAll;
dt=windowsRef2[tableName];
//print(dt<<GetName);
SignalNum=windowsRef3[dt<<GetName];
SignalName=ListSignals[SignalNum];
cstrTable=ListConstraintTables[SignalNum];
dt<<Begin Data Update;
//FILTER ACTIVE MAXIMUM CONSTRAINTS
activeCstr=cstrTable<<Get Rows Where(:Activity=="Active" & :Type=="Max");
for(i=1, i<=NRows(activeCstr),i++,
cstrName=Column(cstrTable, "Name")[activeCstr[i]];
istart=Column(cstrTable, "istart")[activeCstr[i]];
iend=Column(cstrTable, "iend")[activeCstr[i]];
for(m=istart, m<=iend-1, m++,
for(k=1,k<=NumCases,k++,
if(Column(dt,SignalName)[(k-1)*nSamples+m]>Column(dt,cstrName)[(k-1)*nSamples+m],
Column(dt,"Selected_State")[(k-1)*nSamples+m]=1);
);
);
);
//FILTER ACTIVE MINIMUM CONSTRAINTS
activeCstr=cstrTable<<Get Rows Where(:Activity=="Active" & :Type=="Min");
for(i=1, i<=NRows(activeCstr),i++,
cstrName=Column(cstrTable, "Name")[activeCstr[i]];
istart=Column(cstrTable, "istart")[activeCstr[i]];
iend=Column(cstrTable, "iend")[activeCstr[i]];

```

```

for(m=istart, m<=iend-1, m++,
for(k=1,k<=NumCases,k++,
if(Column(dt,SignalName)[(k-1)*nSamples+m]<Column(dt,cstrName)[(k-1)*nSamples+m],
Column(dt,"Selected_State")[(k-1)*nSamples+m]=1);
);
);
);
dt<<End Data Update;
//SELECT POINTS THAT VIOLATE CONSTRAINTS
dt<<Select Where(:Selected_State==1);
dt<<Colors(0);
CurrentDataTable(dt);
);
InvertSelection=expr(
sel1=dt<<Get Rows Where(:Selected_State==1);
sel0=dt<<Get Rows Where(:Selected_State==0 & :Role=="case");
for(i=1, i<=NRows(sel1),i++,
Column(dt,"Selected_State")[sel1[i]]=0;
);
for(i=1, i<=NRows(sel0),i++,
Column(dt,"Selected_State")[sel0[i]]=1;
);
);
HighlightDesirable=expr(//HIGHLIGHT POINTS THAT DO NOT VIOLATE CONSTRAINTS
// Highlight points that violate constraints
FilterSignals;
////////// Highlight entire signals
//DETERMINE LIST OF CASES SELECTED IN SIGNAL PLOT
FindSelectedSignals;
if(NItems(ListSignalsSelected)==0, print("No signal selected"),
//show(ListCases);
//SELECT ENTIRE SIGNALS
dt<<Begin Data Update;
For(i=1, i<=NItems(ListSignalsSelected),i++,
Show(ListSignalsSelected[i]);//TEST
y=dt<<Get Rows Where(:Case==ListSignalsSelected[i]);
Show(NRows(y));//TEST
For(j=1, j<=NRows(y), j++,
//Show(y[j]);//TEST
//dt:Selected_State[y[j]]=1;
//dt:Selected_State[1]=100;//TEST

```

```

Column(dt, "Selected_State")[y[j]]=1;
//rowState(y[j])=color state(37);
);
);
dt<<End Data Update;
//dt<<Select Where(:Selected_State==1);
);
//////////
InvertSelection;
dt<<Select Where(:Selected_State==1);
HighlightSignals;
);
HideSelection=expr();
//-----
// VARIOUS SCRIPTS
//-----
BringWindowsToFront=expr(
for(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
dt=ListSignalTables[SignalNum];
windowsRef1[dt<<GetName]=DynWindow;
DynWindow<<BringWindowToFront;
);
StaticWindow<<BringWindowToFront;
);
//-----
// MORE TEST SCRIPTS
//-----
TestWindowsRef=expr(
for(SignalNum=1, SignalNum<=NItems(ListSignals), SignalNum++,
tableRef=ListSignalTables[SignalNum];
name=tableRef<<GetName;
Show(name);
Show((windowsRef1[name]));
tableRef2={};
tableRef2=(windowsRef2[name]);
Show(tableRef2<<GetName);
Show(eval((windowsRef3[name])));
);
);

```

REFERENCES

- [1] "Ieee guide for aircraft electric systems," Tech. Rep. IEEE Std 128-1976, IEEE, 1976.
- [2] "Systems engineering handbook," Tech. Rep. v2.0, INCOSE (International Council on Systems Engineering), July 2000.
- [3] "Power optimised aircraft," tech. rep., Contract G4RD-2001-00601 under the European Communities 5th Framework Programme for Research, 2001.
- [4] "Vivace: Value improvement through a virtual aeronautical collaborative enterprise," tech. rep., 2007. accessible at <http://www.vivaceproject.com/>.
- [5] AIGNER, W., MIKSCH, S., MÜLLER, W., SCHUMANN, H., and TOMINSKI, C., "Visualizing time-oriented data - a systematic view," *Computers & Graphics*, vol. 31, pp. 401–409, 2007.
- [6] AINTABLIAN, H., FASSBURG, H., and SOENDKER, E., "International space station (iss) truss orbital replaceable unit (oru) overvoltage transient resolution," in *37th Intersociety Energy Conversion Engineering Conference*, 2002.
- [7] ARKADA, A. A. and VANDERHEIDEN, R. H., "Dynamic modeling of high-speed aircraft generators during forced power transfer operation," *Journal of Propulsion and Power*, vol. 11, pp. 1324–1329, November-December 1995.
- [8] ARROWSMITH, D. K. and PLACE, C. M., *An Introduction to Dynamical Systems*. Cambridge University Press, 1990.
- [9] ASCHER, U. M. and RUTH, L., *Computer methods for ordinary differential equations and differential-algebraic equations*. SIAM, 1998.
- [10] BALCHANOS, M., MOON, K., WESTON, N. R., and MAVRIS, D. N., "A dynamic surrogate model technique for power systems modeling and simulation," in *SAE Power Systems Conference*, Georgia Institute of Technology, November 2008.
- [11] BALS, J., HOFER, G., PFEIFFER, A., and SCHALLERT, C., "Virtual iron bird - a multidisciplinary modelling and simulation platform for new aircraft system architectures," 2005.
- [12] BATES, S. J., SIENZ, J., and TOROPOV, V. V., "Formulation of the optimal latin hypercube design of experiments using a permutation genetic algorithm," in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, (Palm Springs, California), April 2004.

- [13] BENAC, C., "Virtual system approach," in *2nd ESA Space Systems Design, Verification & AIT Workshop*, (Noordwijk, The Netherlands), Airbus, 2003.
- [14] BENGIO, Y., *Neural Networks for Speech and Sequence Recognition*. International Thomson Computer Press, 1996.
- [15] BENGIO, Y., SIMARD, P., and FRASCONI, P., "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, March 1994.
- [16] BHANDARI, I., COLET, E., and PARKER, J., "Advanced scout: Data mining and knowledge discovery in nba data," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 121–125, 1997.
- [17] BILLINGS, S. and WEI, H.-L., "A new class of wavelet networks for nonlinear system identification," *IEEE Transactions On Neural Networks*, vol. 16, pp. 862–874, July 2005.
- [18] BODEN, M., "A guide to recurrent neural networks and backpropagation," in *In the Dallas project, SICS Technical Report T2002:03, SICS*, 2002.
- [19] BOLLEN, M. H. J., *Understanding Power Quality Problems: Voltage Sags and Interruptions*. IEEE Press, 2000.
- [20] BRICEÑO, S. I., BUONANNO, M. A., FERNÁNDEZ, I., and MAVRIS, D. N., "A parametric exploration of supersonic business jet concepts utilizing response surfaces," in *AIAA's Aircraft Technology, Integration, and Operations (ATIO)*, 2002.
- [21] BUONANNO, M. A., *A Method for Aircraft Concept Exploration using Multi-criteria Interactive Genetic Algorithms*. PhD thesis, Georgia Institute of Technology, 2005.
- [22] CHEN, Q.-Z., MO, Y.-F., and MENG, G., "Dymola-based modeling of srd in aircraft electrical system," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, pp. 220–227, January 2006.
- [23] CHEN, X., WANG, G.-F., ZHOU, W., ZHANG, Q.-L., and XU, J.-F., "Application of neural networks for integrated circuit modeling," in *Proceedings of the Third International Symposium on Neural Networks, Part III* (WANG, J., YI, Z., ZURADA, J. M., LU, B.-L., and HUJUN, Y., eds.), vol. 3973 of *Advances in Neural Networks*, (Chengdu, China), pp. 1304–1312, Springer, May 2006.
- [24] CHENG, Z., PELLETTIERE, J. A., and WRIGHT, N. L., "Multi-scale validation of automobile impact modeling," in *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, September 2006.
- [25] CLARKE, C. A. and LARSEN, W. E., "Aircraft electromagnetic compatibility," NASA Technical Report NASA-CR-181051, Boeing Commercial Airplane Company and FAA Technical Field Office, June 1987.

- [26] COOPER, G. F., "A simple constraint-based algorithm for efficiently mining observational databases for causal relationships," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 203–224, 1997.
- [27] CORBETT, M. W., LAMM, P. T., MILLER, K. L., WOLFF, J. M., and WALTER, E. A., "Transient analysis of an aircraft/propulsion system with hardware-in-the-loop power extraction," in *43rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, July 2007.
- [28] COSTA, M., PASERO, E., PIGLIONE, F., and RADASANU, D., "Short term load forecasting using a synchronously operated recurrent neural network," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 5, pp. 3478–3482, 1999.
- [29] CRNOSIJA, P., KRISHNAN, R., and BIAZIC, T., "Transient performance based design optimization of pm brushless dc motor drive speed controller," in *IEEE International Symposium on Industrial Electronics*, 2005.
- [30] CROWDER, R. and MAXWELL, C., "Simulation of a prototype electrically powered integrated actuator for civil aircraft," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 211, no. 6, pp. 381–394, 1997.
- [31] CRUSE, H., *Neural Networks as Cybernetic Systems*. Brains, Minds & Media, 2009.
- [32] DABERKOW, D. D., *A Formulation of Metamodel Implementation Processes for Complex Systems Design*. PhD thesis, Georgia Institute of Technology, 2002.
- [33] DABERKOW, D. D. and MAVRIS, D. N., "New approaches to conceptual and preliminary aircraft design: A comparative assessment of a neural network formulation and a response surface methodology," in *World Aviation Conference*, (Anaheim, CA), 1998.
- [34] DAI, R. and JR., J. E. C., "Three-dimensional trajectory optimization in constrained airspace," *Journal of Aircraft*, vol. 46, pp. 627–634, March-April 2009.
- [35] DAUBECHIES, I., *Ten Lectures on Wavelets*. SIAM: Society for Industrial and Applied Mathematics, 1992.
- [36] DE CHAZELLES, P. and FRABOULET, G., "Use of requirement engineering discipline in support of aircraft design," in *1st ESA Space Systems Design, Verification & AIT Workshop*, (Noordwijk, The Netherlands), Airbus, Seditec, 2002.
- [37] DE WECK, O., AGTE, J., SOBIESZCZANSKI-SOBIESKI, J., ARENDSSEN, P., MORRIS, A., and SPIECK, M., "State-of-the-art and future trends in multidisciplinary design optimization," in *48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, April 2007.

- [38] DELAURENTIS, D. A. and MAVRIS, D. N., "Uncertainty modeling and management in multidisciplinary analysis and synthesis," in *38th AIAA Aerospace Sciences Meeting and Exhibit*, (Reno, NV), 12-15 January 2000.
- [39] DEMPSEY, M., "Dymola for multi-engineering modelling and simulation," in *IEE Vehicle Power and Propulsion Conference*, 2006.
- [40] Department Of Defense, *MIL-HDBK-704-7 - Guidance for Test Procedures for Demonstration of Utilization Equipment Compliance to Aircraft Electrical Power Characteristics 270VDC*, April 2004.
- [41] Department Of Defense, *MIL-ST 704F - Aircraft Electric Power Characteristics*, March 2004.
- [42] DIMITRI N. MAVRIS, P. T. B. and WESTON, N. R., "Advanced design of complex systems using the collaborative visualization environment," in *43rd AIAA Aerospace Sciences Meeting and Exhibit*, 2005.
- [43] DOMÉNECH-ASENSI, G., HINOJOSA, J., RUIZ, R., and ÁNGEL DÍAZ-MADRID, J., "Accurate and reusable macromodeling technique using a fuzzy-logic approach," in *IEEE International Symposium on Circuits and Systems*, 2008.
- [44] DUFRESNE, S., *A hierarchical modeling methodology for the definition and selection of requirements*. PhD thesis, Georgia Institute of Technology, 2008.
- [45] DYNASIM, "Dymola." <http://www.dynasim.se/>.
- [46] ELLIS, G., *Control System Design Guide, Third Edition: Using Your Computer to Understand and Diagnose Feedback Controllers*. Elsevier Academic Press, 2004.
- [47] ELMQVIST, H., CELLIER, F. E., and OTTER, M., "Object-oriented modeling of power-electronic circuits using dymola," in *First Joint Conference of International Simulation Societies*, pp. 156–161, August 1994.
- [48] ELMQVIST, N., DRAGICEVIC, P., and FEKETE, J.-D., "Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 1141–1148, November-December 2008.
- [49] ENDER, T., *A Top-Down, Hierarchical, System-of-Systems Approach to the Design of an Air Defense Weapon*. PhD thesis, Georgia Institute of Technology, 2006.
- [50] FAYYAD, U., PIATETSKY-SHAPIO, G., and SMYTH, P., "From data mining to knowledge discovery in databases," *AI Magazine*, vol. 17, pp. 37–54, 1996.
- [51] FILIZADEH, S., GOLE, A. M., WOODFORD, D. A., and IRWIN, G. D., "An optimization-enabled electromagnetic transient simulation-based methodology for hvdc controller design," *IEEE Transactions on Power Delivery*, vol. 22, pp. 2559–2566, 2007.

- [52] FÉLIX, M. and ROUTEX, J.-Y., “A copper bird for aircraft equipment systems integration and electrical network characterization,” in *45th AIAA Aerospace Sciences Meeting and Exhibit*, (Reno, Nevada), January 2007.
- [53] FU, M. C., “Optimization via simulation: A review,” *Annals of Operation Research*, vol. 53, pp. 199–248, 1994.
- [54] FU, M. C., “Optimization for simulation: Theory vs. practice,” *INFORMS Journal on Computing*, vol. 14, pp. 192–215, Summer 2002.
- [55] GANTHONY, D., SCHOFIELD, N., and BINGHAM, C. M., “Conditioning of aircraft flight control surface actuation loads,” in *16th International Conference on Electrical Machines*, 2004.
- [56] GILES, L., LAWRENCE, S., and TSOI, A. C., “Rule inference for financial prediction using recurrent neural networks,” in *IEEE Conference on Computational Intelligence for Financial Engineering*, p. 253, IEEE Press, 1997.
- [57] GILL, P., MURRAY, W., and M.H.WRIGHT, *Practical Optimization*. London: Academic Press, 1981.
- [58] GILL, P., MURRAY, W., and M.H.WRIGHT, *Numerical Linear Algebra and Optimization*, vol. 1. Addison Wesley, 1991.
- [59] GIUNTA, A. A., WOJTKIEWICZ-JR., S. F., and ELDRED, M. S., “Overview of modern design of experiments methods for computational simulations,” in *41st AIAA Aerospace Sciences Meeting and Exhibit*, (Reno, Nevada), January 2003.
- [60] GREENWOOD, A., *Electrical Transients in Power Systems*. John Wiley & Sons, second ed., 1991.
- [61] HAN, J. and KAMBER, M., *Data Mining: Concepts and Techniques*. Academic Press, 2001.
- [62] HORNE, G. E. and MEYER, T. E., “Data farming: discovering surprise,” in *Proceedings of the 2004 Winter Simulation Conference* (INGALLS, R. G., ROSETTI, M. D., SMITH, J. S., and PETERS, B. A., eds.), pp. 807–813, 2004.
- [63] HORNIK, K., “Approximation capabilities of multilayer feedforward networks,” *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.
- [64] HORNIK, K., STINCHCOMBE, M., and WHITE, H., “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 2, pp. 359–366, 1989.
- [65] HOWSE, M., “All electric aircraft,” *Power Engineer*, vol. 17, pp. 35–37, August–September 2003.
- [66] HU, Y. H. and HWANG, J.-N., eds., *Handbook of Neural Network Signal Processing*. CRC Press, 2002.

- [67] HUA, L. and FENG, F., "Simulation study of power quality disturbance in distributed power system using complex wavelet network," in *The Eighth International Conference on Electronic Measurement and Instruments*, pp. 3–426–3–429, IEEE, 2007.
- [68] HULL, D. G., *Optimal Control Theory for Applications*. Springer-Verlag, 2003.
- [69] JAWERTH, B. and SWELDENS, W., "An overview of wavelet based multiresolution analyses," *SIAM Review*, vol. 36, pp. 377–412, 1993.
- [70] JEONG, D.-S., CHOI, K.-J., WOO, H.-W., and KIM, J.-G., "Controller design of missile actuator using dsp," in *International Conference on Control, Automation and Systems*, October 2007.
- [71] JIAO, L., PAN, J., and FANG, Y., "Multiwavelet neural network and its approximation properties," *IEEE Transactions on Neural Networks*, vol. 12, pp. 1060–1066, September 2001.
- [72] JIN, L., NIKIFORUK, P. N., and GUPTA, M. M., "Approximation capability of feedforward and recurrent neural networks," in *Intelligent Control Systems: Concepts and Applications* (GUPTA, M. M. and SINHA, N. K., eds.), p. 235, IEEE Press, 1995.
- [73] JIN, Q., WONG, K., and LUO, Z., "Wavelet function, scaling function and discrete wavelet transform," in *Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis*, pp. 409–412, 1994.
- [74] JOHNSON, C. and SCHUTTE, J., "Basic regression analysis for integrated neural networks (brainn) documentation," tech. rep., Aerospace Systems Design Laboratory, School of Aerospace Engineering, Georgia Institute of Technology, 2007.
- [75] JOUANNET, C. and KRUS, P., "Direct simulation based optimization for aircraft design including systems," in *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, (Portsmouth, Virginia), 2006.
- [76] JR., J. J. L., PRABHAT, FORSBERG, A. S., LAIDLAW, D. H., and VAN DAM, A., *Trends in Interactive Visualization: State-of-the-Art Survey*, ch. 10: Virtual Reality-Based Interactive Scientific Visualization Environments, pp. 225–250. Springer-Verlag, 2009.
- [77] JUDITSKY, A., JUDITSKY, A., ZHANG, Q., ZHANG, Q., DELYON, B., DELYON, B., Y. GLORENNEC, P., Y. GLORENNEC, P., and BENVENISTE, A., "Wavelets in identification - wavelets, splines, neurons, fuzzies: How good for identification ?," tech. rep., IRISA, 1994.
- [78] KANKAM, D. and ELBULUK, M., "A survey of power electronics applications in aerospace technologies," Tech. Rep. NASA/TM-2001-211298, NASA Glenn, University of Akron, 2001.

- [79] KAPUR, S., LONG, D. E., and ROYCHOWDHURY, J., "Efficient time-domain simulation of frequency-dependent elements," in *International Conference on Computer-Aided Design (ICCAD '96)*, 1996.
- [80] KAZIBWE, W. E. and SENDAULA, M. H., *Electric Power Quality Control Techniques*. Van Nostrand Reinhold, 1993.
- [81] KELEMEN, A. and IMECS, M., "Mathematical model and digital simulation of a current-fed induction motor system," *Mathematical Modelling*, vol. 8, pp. 550–555, 1987.
- [82] KHANNA, V. K., *The Insulated Gate Bipolar Transistor: IGBT Theory and Design*. Wiley-IEEE Press, 2003.
- [83] KIRBY, M. R., *A Methodology for Technology Identification, Evaluation, and Selection in Conceptual and Preliminary Aircraft Design*. PhD thesis, Georgia Institute of Technology, 2001.
- [84] KIRK, D. E., *Optimal Control Theory: An Introduction*. Dover Publications, 2004.
- [85] KOTSIANTIS, S. B., "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.
- [86] KRUS, P., "Systems engineering in aircraft system design," in *11th Annual International Symposium of The International Council On Systems Engineering*, (Melbourne, Australia), INCOSE, 2001.
- [87] KTONAS, P. Y. and PAPP, N., "Instantaneous envelope and phase extraction from real signals: Theory, implementation, and application to eeg analysis," *Signal Processing*, vol. 2, pp. 373–385, October 1980.
- [88] KUGARAJAH, T. and ZHANG, Q., "Multidimensional wavelet frames," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1552–1556, 1995.
- [89] KULAKOWSKI, B. T., GARDNER, J. F., and SHEARER, J. L., *Dynamic Modeling and Control of Engineering Systems*. Cambridge University Press, third ed., 2007.
- [90] KUNDERT, K. S., "Introduction to rf simulation and its application," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 1298–1319, September 1999.
- [91] KUNDUR, P., *Power System Stability and Control*. McGraw-Hill, 1994.
- [92] LAW, A. M. and KELTON, W. D., *Simulation Modeling and Analysis*. McGraw-Hill, 1991.
- [93] LEE, C.-H., WANG, Y.-J., and HUANG, W.-L., "A literature survey of wavelets in power engineering applications," in *Proc. Natl. Sci. Council. ROC(A)*, vol. 24, pp. 249–258, 2000.

- [94] LEE, H., PARK, Y., MEHROTRA, K., MOHAN, C., and RANKA, S., "Nonlinear system identification using recurrent networks," in *1991 IEEE International Joint Conference on Neural Networks*, 1991.
- [95] LIAO, T. W., "Clustering of time series data - a survey," *Pattern Recognition*, vol. 38, pp. 1857–1874, 2005.
- [96] LIEFVENDAHL, M. and STOCKI, R., "A study on algorithms for optimization of latin hypercubes," *Journal of Statistical Planning and Inference*, vol. 136, pp. 3231–3247, September 2006.
- [97] LIM, D., *A Systematic Approach to Design for Lifelong Aircraft Evolution*. PhD thesis, Georgia Institute of Technology, 2009.
- [98] LIN, J., KEOGH, E., LONARDI, S., LANKFORD, J. P., and NYSTROM, D. M., "Visually mining and monitoring massive time series," in *In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 460–469, ACM Press, 2004.
- [99] LIU, G. P., BILLINGS, S. A., and KADIRKAMANATHAN, V., "Nonlinear system identification using wavelet networks," in *UKACC International Conference on Control*, no. 455, pp. 1248–1253, IEE, September 1998.
- [100] LJUNG, L., *System Identification: Theory for the User*. Prentice Hall Professional Technical Reference, second ed., 1998.
- [101] LUO, F.-L. and UNBEHAUEN, R., *Applied Neural Networks for Signal Processing*. Cambridge University Press, 1997.
- [102] MALLAT, S., *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 2008.
- [103] MALLAT, S. G., "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 674–693, July 1989.
- [104] MARTIN, J. N., "Overview of the eia 632 standard: processes for engineering a system," in *17th AIAA/IEEE/SAE Digital Avionics Systems Conference*, 1998.
- [105] MATHWORKS, *MATLAB, Release R2009a*. <http://www.mathworks.com/>, 2009.
- [106] MATHWORKS, *Simulink, Version 7.3*. <http://www.mathworks.com/>, 2009.
- [107] MAVRIS, D. N., BANDTE, O., and DELAURENTIS, D. A., "Robust design simulation: A probabilistic approach to multidisciplinary design," *Journal of Aircraft*, vol. 36, pp. 298–307, January-February 1999.
- [108] MEDSKER, L. R. and JAIN, L. C., eds., *Recurrent Neural Networks: Design and Applications*. CRC Press, 2000.

- [109] MEIJER, P. B. L., *Neural Network Applications in Device and Subcircuit Modelling for Circuit Simulation*. PhD thesis, Technical University of Eindhoven, 1996.
- [110] METROPOLIS, N., "The beginning of the monte carlo method," *Los Alamos Science*, vol. Special Issue, pp. 125–130, 1987.
- [111] MICHEL, A. N. and LIU, D., *Qualitative Analysis and Synthesis of Recurrent Neural Networks*. Marcel Dekker, 2002.
- [112] MIRI, S. M. and KEYHANI, A., "Modeling of a three-phase power conditioning system and improving its transient behavior following a heavy load drop," in *Industry Applications Society Annual Meeting, Conference Record of the 1989 IEEE*, pp. 1793–1800, 1986.
- [113] MISRIKHANOV, A. M., "Wavelet transform methods: Application in electroenergetics," *Automation and Remote Control*, vol. 67, no. 5, pp. 682–697, 2006.
- [114] MITCHELL, M., *An Introduction to Genetic Algorithms*. MIT Press, 1998.
- [115] MOHAN, N., UNDELAND, T. M., and ROBBINS, W. P., *Power Electronics: Converters, Applications, and Design*. John Wiley & Sons, 2002.
- [116] MOIR, I., "The all-electric aircraft - major challenges," in *IEE Colloquium on All Electric Aircraft*, June 1998.
- [117] MONTGOMERY, D. C., *Design and Analysis of Experiments*. John Wiley & Sons, fourth ed., 1997.
- [118] MYERS, R. H., MONTGOMERY, D. C., and ANDERSON-COOK, C. M., *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. John Wiley & Sons, third ed., 2009.
- [119] NAGRATH, J., *Control Systems Engineering*. New Age International Publishers, 2009.
- [120] NAIDU, D. S., *Optimal Control Systems*. CRC Press, 2003.
- [121] NARENDRA, K. S. and PARTHASARATHY, K., "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 4–27, March 1990.
- [122] NAZARUDDIN, Y. Y. and YULIATI, "Wavenet based modeling of vehicle suspension system," in *32nd IEEE Annual Conference on Industrial Electronics*, pp. 144–149, November 2006.
- [123] NISE, N. S., *Control Systems Engineering*. Wiley, fifth ed., 2007.
- [124] PARK, J.-S., "Optimal latin-hypercube designs for computer experiments," *Journal of Statistical Planning and Inference*, vol. 39, pp. 95–111, April 1994.

- [125] PATEL, A. V., PATEL, V. V., DEODHARE, G., and CHETTY, S., "Flight control system clearance using static tests at iron bird," in *AIAA Guidance, Navigation, and Control Conference and Exhibit* (AIAA, ed.), (Keystone, Colorado), August 2006.
- [126] PEARLMUTTER, B. A., "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1212–1228, September 1995.
- [127] PHAN, L. L., MAVRIS, D. N., CHARRIER, J.-J., THALIN, P., and GARCIA, E., "Parametric modeling of an electrical test rig for power optimized aircraft architectures," in *The 26th Congress of International Council of the Aeronautical Sciences (ICAS), including the 8th AIAA Aviation Technology, Integration, and Operations*, September 2008.
- [128] PLEBE, A., ANILE, A. M., and RINAUDO, S., "Neural networks in circuit simulators," in *Proceedings of the 11th International Conference on Artificial Neural Networks*, (Vienna, Austria), pp. 699–705, Springer, August 2001.
- [129] RAYMER, D. P., *Aircraft Design: A Conceptual Approach*. AIAA Education Series, 1992.
- [130] ROJAS, R., *Neural Networks - A Systematic Introduction*. Springer-Verlag, 1996.
- [131] SAFFER, J. D., BURNETT, V. L., CHEN, G., CHEN, G., and VAN DER SPEK, P., "Visual analytics in the pharmaceutical industry," *IEEE Computer Graphics and Applications*, vol. 24, pp. 10–15, September-October 2004.
- [132] SAS Institute, *JMP*. <http://www.jmp.com/>.
- [133] SCHETZEN, M., *Linear Time-Invariant Systems*. Wiley-IEEE Press, 2002.
- [134] SHEEN, Y.-T. and HUNG, C.-K., "Constructing a wavelet-based envelope function for vibration signal analysis," *Mechanical Systems and Signal Processing*, vol. 18, pp. 119–126, 2004.
- [135] SHENG-FU LIANG, A. W. S. and LIN, C.-T., "A new recurrent-network-based music synthesis method for chinese plucked- string instruments - pipa and qin," in *Proceedings of the International Joint Conference on Neural Networks*, 1999.
- [136] SHENKMAN, A. L., *Transient Analysis of Electric Power Circuits Handbook*. Springer, 2005.
- [137] SINGH, G., "Visual fusion," *IEEE Computer Graphics and Applications*, vol. 24, pp. 4–5, 2004.
- [138] SJÖBERG, J., "Some examples of identification with neural networks," tech. rep., 1994.

- [139] SJÖBERG, J., ZHANG, Q., LJUNG, L., BENVENISTE, A., DEYLON, B., YVES GLORENNEC, P., HJALMARSSON, H., and JUDITSKY, A., “Nonlinear black-box modeling in system identification: a unified overview,” *Automatica*, vol. 31, pp. 1691–1724, 1995.
- [140] SMITH, M., *Neural Networks for Statistical Modeling*. Van Nostrand Reinhold, 1993.
- [141] SOLTANI, S., “On the use of the wavelet decomposition for time series prediction,” *Neurocomputing*, vol. 48, pp. 267–277, 2002.
- [142] SUI, W., *Time-domain computer analysis of nonlinear hybrid systems*. CRC Press, 2001.
- [143] TATIZAWA, H., BURANI, G. F., and OBASE, P. F., “Application of computer simulation for the design of a new high voltage transducer, aiming to high voltage measurements at field, for dc measurements and power quality studies,” *WSEAS Transactions on Systems*, vol. 7, no. 5, pp. 580–589, 2008.
- [144] TAUBMAN, D. S. and MARCELLIN, M. W., eds., *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Springer, 2002.
- [145] TEOLIS, A., *Computational Signal Processing with Wavelets*. Birkhäuser Boston, 1998.
- [146] THOMAS, J. and COOK, K., eds., *Illuminating the Path: The Research and Development Agenda for Virtual Analytics*. The National Visualization and Analytics Center, 2004.
- [147] TIIRA, T., “Detecting teleseismic events using artificial neural networks,” *Computers and Geosciences*, vol. 25, pp. 929–938, September 1999.
- [148] TISCHLER, M. B. and REMPLE, R. K., *Aircraft and Rotorcraft System Identification: Engineering Methods with Flight-Test Examples*. American Institute of Aeronautics and Astronautics, 2006.
- [149] TOUMAZOU, C. and BARRY, G., “Intuitive analogue circuit design,” *Electronics and Communication Engineering Journal*, vol. 9, pp. 231–239, October 1997.
- [150] TRIKHA, A. K., “Reduction of hydraulic system pressure spikes through limiting of servovalve current change rate,” in *AIAA and SAE, 1998 World Aviation Conference*, Boeing Co., 1998.
- [151] VANDERPLAATS, G. N., *Numerical Optimization Techniques for Engineering Design*. Vanderplaats Research and Development, third ed., 1999.
- [152] WANG, Z. and BOVIK, A. C., “Mean squared error: Love it or leave it? a new look at signal fidelity measures,” *IEEE Signal Processing Magazine*, vol. 26, pp. 98–117, January 2009.

- [153] WASSINK, I., KULYK, O., VAN DIJK, B., VAN DER VEER, G., and VAN DER VET, P., *Trends in Interactive Visualization: State-of-the-Art Survey*, ch. 8: Applying a User-Centered Approach to Interactive Visualization Design, pp. 175–199. Springer-Verlag, 2009.
- [154] WEBER, M., ALEXA, M., and MÜLLER, W., “Visualizing time-series on spirals,” in *IEEE Symposium on Information Visualization (InfoVis 2001)*, pp. 7–14, 2001.
- [155] WIJK, J. J. V. and VAN SELOW, E. R., “Cluster and calendar based visualization of time series data,” in *In INFOVIS 99: Proceedings of the 1999 IEEE Symposium on Information Visualization*, pp. 4–9, IEEE Computer Society, 1999.
- [156] WONG, P. C., FOOTE, H., KAO, D. L., LEUNG, R., and THOMAS, J., “Multivariate visualization with data fusion,” *Information Visualization*, vol. 1, pp. 182–193, December 2002.
- [157] WONG, P. C. and THOMAS, J., “Visual analytics,” *IEEE Computer Graphics and Applications*, vol. 24, pp. 20–21, September-October 2004.
- [158] WOODS, E. J., “Aircraft electrical system computer simulation,” in *Energy Conversion Engineering Conference*, August 1990.
- [159] WYSS, G. D., JORGENSEN, K. H., IMAN, R. L., SHORTENCARIER, M. J., and DAVENPORT, J. M., “A user’s guide to lhs: Sandia’s latin hypercube sampling software,” 1998.
- [160] ZHANG, J., WALTER, G., MIAO, Y., and LEE, W. N. W., “Wavelet neural networks for function learning,” *IEEE Transactions On Signal Processing*, vol. 43, pp. 1485–1497, June 1995.
- [161] ZHANG, Q., “Using wavelet network in nonparametric estimation,” *IEEE Transactions on Neural Networks*, vol. 8, pp. 227–236, March 1994.
- [162] ZHANG, Q. and BENVENISTE, A., “Wavelet networks,” *IEEE Transactions on Neural Networks*, vol. 3, pp. 889–898, November 1992.
- [163] ZUDILOVA-SEINSTRA, E., ADRIAANSEN, T., and VAN LIERE, R., *Trends in Interactive Visualization: State-of-the-Art Survey*, ch. 1: Overview of interactive visualization, pp. 3–18. Springer-Verlag, 2009.

VITA

Léon Phan was born in 1980 in Aubervilliers, near Paris, in France. After high school and three years of intensive “preparation classes for national competitive exams” in Paris, he entered the Ecole Nationale Supérieure de l’Aéronautique et de l’Espace (SUPAERO) in Toulouse in 2000. In 2002, he entered the School of Aerospace Engineering at the Georgia Institute of Technology for a Master of Science within the Aerospace Systems Design Laboratory (ASDL), for a dual-degree curriculum. In December 2003, he thus concurrently obtained his Master of Science from Georgia Tech and his French engineering degree from SUPAERO.

While pursuing his Ph.D. degree at ASDL with Dr. Dimitri Mavris, his research first focused on the topic of aircraft system and subsystem architectures. In June 2006, he spent three months at Hispano-Suiza (Safran Group) to work on the COPPER Bird[®], an electrical test rig developed in the context of the Power Optimised Aircraft Program (POA), a research effort funded by the European Union on the topic of More-Electric Aircraft. He developed surrogate models of the test rig simulating the behavior of an aircraft electrical network. It was this work that set the stage for his thesis work on the efficient integration of transient constraints in the design of aircraft dynamic systems.